深入学习webpack

1. 课前准备

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

2. 课堂主题

3. 课堂目标

1. 手写webpack原理
2. 定制自己的lodaer和plugin

4. 知识点

###

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

kkbpack内部npm init 新建src/index.js

```
#! /usr/bin/env node
// 声明当前问件事node的文件

console.log('俺是kkbpack')
```

开发命令行工具的时候，需要package.json厉声明bin字段，告诉npm，当前这个项目作为命令行使用的时候，执行哪个文件

```
  "bin":{
    "kkbpack":"./src/index.js"
  },
```

执行npm link，把当前kkb链接到本地全局

```
up to date in 10.215s
/usr/local/bin/kkbpack -> /usr/local/lib/node_modules/kkbpack/src/index.js
/usr/local/lib/node_modules/kkbpack -> /Users/woniuppp/mygithub/webpack-lesson/kkbpack
```

这样就全局安装了Kkbpack，并且修改实时生肖，我们回到02目录，执行kkbpack

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

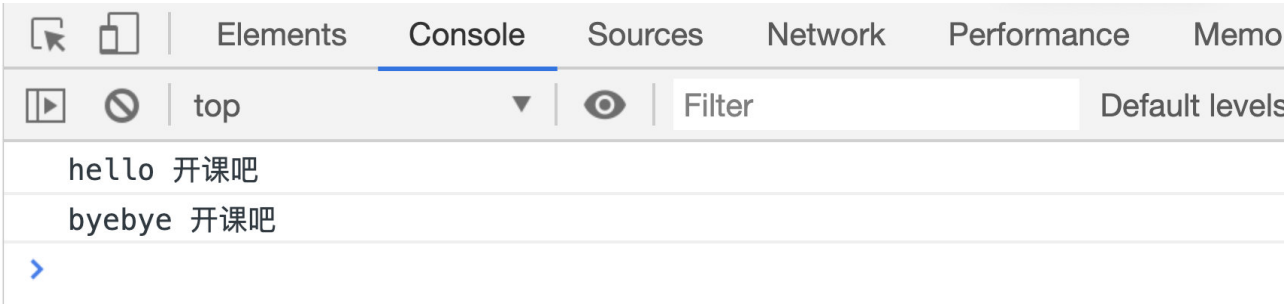
打开dist/kkb.js

```
// 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>
```



bingo 回顾下代码

```
#! /usr/bin/env node
const path = require('path')
const fs = require('fs')
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
        // 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))
        // this.modules[name] = code
        this.modules[name] = `function(module,exports, __kkbpack_require__){
            eval(\`${code}\`)
        }`
        // 递归获取依赖
        deps.forEach(dep=>{
```

```
            this.createModule(path.join(this.root,dep), './'+dep)
        })
    }
    parse(code, parent){
        let deps = []
        // 识别 require('xx')
        var r = /require\((.*)\)/g;
        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__("./${retPath}")`
        })
        return {code, deps}
    }
    generateModuleStr(){
        // 生成model字符串
        let fnTemp = ""
        Object.keys(this.modules).forEach(name=>{
            fnTemp += `"${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()

    }

}

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

console.log('------\n')
```

loader

新的文件类型做特殊处理，就需要loader出场了，我们先来整一个style-loader

新建css

查看打包后的js

```
sayHi('开课吧')`)
        },"./src/index.css":function(module,exports, __kkbpack_require__){
            eval(`h1{
    color:red;
}

p{
    color:blue;
}`)
```

明显要报错，所以我们写一个style-loader，把这些css，放在style标签里就可以啦, 所以的lodaer就是一个函数，传递源代码，返回转换后的代码即可

```
<div>
    <h1>开课吧</h1>
    <p>哈哈</p>
</div>
```

kkb-style-loader

```
module.exports = (code)=>{
    let style = `
        let style = document.createElement('style')
        style.innerHTML = "${code.split('\n').join('')}"
        document.head.appendChild(style)
    `
    return style
}
```

配置

```
module.exports = {
    output:{
        filename:'kkb.js'
    },
    module:{
        rules:[
            {test:/.css$/ , use:['kkb-loader/style-loader',]},
        ]
    }
}
```

let fileContent = this.getCode(modulePath)

```
getCode(modulePath){
    let content = fs.readFileSync(modulePath, 'utf-8')
    this.config.module.rules.forEach(r=>{
        // 匹配rules
        if(r.test.test(modulePath)){
            // 匹配到了，调用laoder
            const loader = require(r.use[0])
            content = loader(content)
        }
    })
    return content
}
```

如果notfound 记得npm link kkb-loader

file-loader

支持图片loader 涉及到md5计算，图片copy

```
let crypto = require('crypto');
let fs = require('fs')
let md5 = crypto.createHash('md5');
function loader(code, name, fullpath){
    // 内容，文件相对路径，全路径
    const output = this.config.output.path
    let ext = name.split('.').pop()
    const filename = md5.update(code).digest('hex')+'.'+ext

    // var base64str = new Buffer.from(code).toString('base64')
    fs.copyFileSync(fullpath, `${output}/${filename}`)
    return `module.exports="${filename}"`
}
```

```
module.exports = loader

// index.js
content = loader.call(this,content,name,modulePath)
```

plugins

webpack可以实现loader所不能完成的复杂功能，使用plugin丰富的自定义API以及生命周期事件，可以控制webpack编译流程的每个环节，实现对webpack的自定义功能扩展。

plugin是一个具有 `apply` 方法的 js对象。 apply方法会被 webpack的 `compiler` （编译器）对象调用，并且 `compiler` 对象可在整个 `compilation` （编译）生命周期内访问。

webpack内部使用tabable在每个流程内部挂载钩子，plugin就可以在不同的阶段，去做额外的一些处理

webpack事件钩子https://webpack.js.org/api/plugins/compiler/#event-hooks

```
class Banner{
    constructor(content){
        this.content = content
    }
    apply(compiler){
        console.log('plugin执行拉')
    }
}

module.exports = Banner
```

```
 event.js
// 建议事件通知
class EventBus{
    constructor(){
        this.callbacks = []
    }
    on(callback){
        this.callbacks.push(callback)
    }
    emit(){
        this.callbacks.forEach(cb=>cb())
    }
}
```

```
module.exports = EventBus

        this.hooks = {
            // 编译
            emit: new EventBus(),
            run: new EventBus(),
            done: new EventBus()
        }
        this.handlePlugins()

    }
    handlePlugins(){
        let plugins = this.config.plugins || []
        plugins.forEach(plugin=>{
            plugin.apply(this)
        })
    }
```

```
    generateFile(){
        // console.log(this.modules, this.entry)
        // console.log(123,this.root)
        let template = fs.readFileSync(path.resolve(__dirname,'./template.js'), 'utf-
8')
        this.template = template.replace('__entry__',this.entry)
                                .replace('__content__', this.generateModuleStr())
        this.hooks.emit.emit()
        fs.writeFileSync('./dist/'+this.config.output.filename, this.template)
    }
```

```
class Banner{
    constructor(content){
        this.content = content
    }

    apply(compiler){
        console.log('plugin执行拉')
        compiler.hooks.run.on(()=>{
            console.log('任务开始跑了')
        })
        compiler.hooks.emit.on(()=>{
            console.log(compiler.template)
            compiler.template = `
//      ┌ ┗━━━┛ ┐
//      |       |
//      |   -   |
```

```
//      ┃ ┳   ┳ ┃
//      ┃       ┃
//      ┃   ┻   ┃
//      ┃       ┃
//      ┗┓     ┏┛
//        ┃     ┃   神兽保佑
//        ┃     ┃   代码无BUG!
//        ┃     ┗━━━━━┓
//        ┃           ┣┓
//        ┃           ┏┛
//        ┗┓┓┏━┳┓┏┛
//         ┃┫┫ ┃┫┫
//         ┗┻┛ ┗┻┛
            `+compiler.template
            // compiler.template = `/** ${this.content} */\n`+compiler.template
            console.log('完成啦 还没写文件')
        })
        compiler.hooks.done.on(()=>{
            // consol
            console.log('写文件结束')
        })
    }
}

module.exports = Banner
```

5. 扩展

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

6. 总结

深入学习webpack

1. 课前准备
2. 课堂主题
3. 课堂目标
4. 知识点
    - kkbpack
    - webpack编译后代码
    - npm link
    - 读取配置文件
    - 读取入口文件
    - 解析源码
    - 依赖列表
    - 生成文件
    - build
    - loader

7. 预告

性能优化

大圣老师