**webpack配置**

* **Step1:** 创建项目文件夹, 名字自起, 但路径名不要包含中文, 以及项目名也不要叫”webpack”, 并且不要包含大写字母.  
  例:
* **Step2:** 接下来创建并编写配置文件. 首先我们先介绍几个配置文件的参数.   
  + entry： 是 页面入口文件配置 （html文件引入唯一的js 文件）
  + output：对应输出项配置   
    - path ：入口文件最终要输出到哪里，
    - filename：输出文件的名称
    - publicPath：公共资源路径
* **Step3:** 在你的项目目录下创建**webpack.config.js**配置文件, 通过这个文件进行webpack的配置, 并且还要创建一些路径保存基本文件. 例如:   
    
  ***src文件夹***
* **Step4:**在src的js下创建一个入口文件, 我创建的叫做entry.js, 在项目目录再创建一个index.html用来调试使用. 编写**webpack.config.js**文件,
* //webpack.config.js
* module.exports = {
* entry : './src/js/entry.js',//入口文件
* output : {//输出文件
* filename : 'index.js',//输出文件名
* path : \_\_dirname + '/out'//输出文件路径
* },

}

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8

我们随便在index.html和入口文件entry.js写点什么看看是否成功配置,

//index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>ss</title>

</head>

<body>

111

<script src="./out/index.js"></script>//注意在这里引入的是打包完成的js文件

</body>

</html>

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11
  + 12

//entry.js

console.log('1234');

* + 1
  + 2

之后使用Git Bash here 或者 cmd cd命令使目录为当前项目目录, 输入**webpack**或者**webpack -w**命令, 查看index.html是否成功console出1234;

webpack 和 webpack -w 区别   
webpack -w可以对项目打包并且实时监控, 当前配置文件下的文件如果发生更改时重新打包, 但如果webpack的配置文件即webpack.config.js更改时还是要通过webpack进行打包.(退出webpack -w 操作 ctrl+c)

**webpack loader加载器**

接下来我们继续配置loader, 通过加载器处理文件：比如 sass less 等, 告知 webpack 每一种文件都需要使用什么加载器来处理。

* **Step1:**为了方便我们先统一把所有的包都先下载下来, 下面再慢慢解释.
* npm install babel-loader babel babel-core css-loader style-loader url-loader file-loader less-loader less --save-dev
  + 1
  + 2
* **Step2:**下载完成后, 我们修改webpack.config.js文件, 将加载器引入.
* module.exports = {
* entry : './src/js/entry.js',
* output : {
* filename : 'index.js',
* path : \_\_dirname + '/out'
* },
* module : {
* rules: [
* {test: /.js$/, use: ['babel-loader']},
* {test: /.css$/, use: ['style-loader', 'css-loader']},/\*解析css, 并把css添加到html的style标签里\*/
* //{test: /.css$/, use: ExtractTextPlugin.extract({fallback: 'style-loader',use: 'css-loader'})},/\*解析css, 并把css变成文件通过link标签引入\*/
* {test: /.(jpg|png|gif|svg)$/, use: ['url-loader?limit=8192&name=./[name].[ext]']},/\*解析图片\*/
* {test: /.less$/, use: ['style-loader', 'css-loader', 'less-loader']}/\*解析less, 把less解析成浏览器可以识别的css语言\*/
* ]
* },

}

* **Step3:**接下来我们先测试css, 我们在项目文件夹下的src文件夹下创建index.css. 随便写一点属性.
* //index.css
* .demo1 {
* width: 100px;
* height: 100px;
* background: red;
* }
* .demo2 {
* width: 200px;
* height: 200px;
* background: orange;

}

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11

//index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>text</title>

</head>

<body>

<div class="demo1"></div>

<div class="demo2"></div>

<script src="./out/index.js"></script>

</body>

</html>

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11
  + 12
  + 13

因为在webpack中所有文件都是模块, 所以必须要将css引入. 在刚刚的entry.js中添加如下代码.

//entry.js

require('../css/index.css');//引入css文件

console.log("1234");

* + 1
  + 2
  + 3

打包webpack一下看看效果 

* **Step4:**当有多个js文件时, 如何进行引入呢? 接下来我们做一个简单小功能来说明这个问题, 让我们点击方块的时候, 方块变大.

接下来在src的js文件夹下创建一个基本的工具tool.js **(很多小的问题都被我扩大化了, 只是为了说明问题不一定适用)**

//tool.js

var tool = {//获取DOM元素

getDom: function(className) {

return document.getElementsByClassName(className)[0];

}

}

module.exports = tool;//模块出口

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8

src的js下创建一个demo1.js文件, demo2.js同理

var obj = require('./tool.js');

var demo1 = {

init: function() {

this.bindEvent();

},

bindEvent: function() {

//var demo1 = document.getElementsByClassName('demo1')[0];

var demo1 = obj.getDom('demo1');

demo1.onclick = this.changeStyle;

},

changeStyle: function() {

this.style.width = '200px';

this.style.height = '200px';

this.style.background = 'green';

console.log('1');

}

}

module.exports = demo1;

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11
  + 12
  + 13
  + 14
  + 15
  + 16
  + 17
  + 18
  + 19

修改入口文件entry.js

require('../css/index.css');

var demo1 = require('../js/demo1.js');

var demo2 = require('../js/demo2.js');

demo1.init();

demo2.init();

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7

webpack一下, 看看效果 

**关于图片的打包**

* **Step1:**在img文件夹下随便找一个小一点的图片放进去.
* **Step2:**修改entry.js
* require('../css/index.css');
* var demo1 = require('../js/demo1.js');
* var demo2 = require('../js/demo2.js');
* demo1.init();
* demo2.init();
* var oImg = new Image();
* oImg.src = require('../img/1.gif');//当成模块引入图片

document.body.appendChild(oImg);

由于我们引入的是静态资源, 在webpack.config.js中修改一下

module.exports = {

entry : './src/js/entry.js',

output : {

filename : 'index.js',

publicPath: \_\_dirname + '/out',//添加静态资源, 否则会出现路径错误

path : \_\_dirname + '/out'

},

module : {

rules: [

{test: /.js$/, use: ['babel-loader']},

{test: /.css$/, use: ['style-loader', 'css-loader']},/\*解析css, 并把css添加到html的style标签里\*/

//{test: /.css$/, use: ExtractTextPlugin.extract({fallback: 'style-loader',use: 'css-loader'})},/\*解析css, 并把css变成文件通过link标签引入\*/

{test: /.(jpg|png|gif|svg)$/, use: ['url-loader?limit=8192&name=./[name].[ext]']},/\*解析图片\*/

{test: /.less$/, use: ['style-loader', 'css-loader', 'less-loader']}/\*解析less, 把less解析成浏览器可以识别的css语言\*/

]

},

}

大家自己webpack看看效果