**第一章：基本构建**

**第一步：项目初始化**

前提：已全局安装了node.js

1. 新建一个文件夹
2. 在文件夹内开打cmd命令行输入npm init –y （生成一个package.json文件，-y是不用一步一步询问，直接生成）
3. 然后执行npm install --save-dev webpack 在开发环境中安装webpack，成功后可用webpack –v查询，并且查看package.json文件中的”devDependencies”:{} 确保里面有webpack和版本号记录。（注意：今后所有的--save-dev安装后都要查看安装的东西是否已经在package.json文件中的”devDependencies”:{}中记录。所有的--save安装后，都要查看package.json文件中的”dependencies”:{}中是否已有记录）
4. 创建src目录用于存放开发环境中用的文件。
5. 创建dist目录用于存放打包后生成的生产环境用的文件。
6. 在src文件中建立一个entry.js作为主入口文件。再建立一个index.html作为静态模板。

**第二步：创建webpack的配置文件**

在根目录下创建一个webpack.config.js文件

webpack.config.js的结构如下：

module.exports={

    //入口文件的配置项

    entry:{},

    //出口文件的配置项

    output:{},

    //模块：例如解读CSS,图片如何转换，压缩

    module:{},

    //插件，用于生产模版和各项功能

    plugins:[],

    //配置webpack开发服务功能

    devServer:{}

}

说明：

* entry：配置入口文件的地址，可以是单一入口，也可以是多入口。
* output：配置出口文件的地址，在webpack2.X版本后，支持多出口配置。
* module：配置模块，主要是解析CSS和图片转换压缩等功能。
* plugins：配置插件，根据你的需要配置不同功能的插件。
* devServer：配置开发服务功能，后期我们会详细讲解。

**第三步：对webpack.config.js进行基本配置**

基本配置如下（配置完即可实现简单的打包操作）：

3.1配置入口和出口

需要在webpack.config.js文件中引入const path=require(‘path’)

然后在entry和output属性中进行配置

3.2配置一个开发环境的服务功能

需要先安装 npm install webpack-dev-server --save-dev

然后在webpack.config.js文件的devServer属性里进行配置

3.3 配置html-webpack-plugin插件（用于管理入口文件的模板index.html文件）

需要先安装npm install --save-dev html-webpack-plugin

然后在webpack.config.js文件中引入 const htmlPlugin=require(‘html-webpack-plugin’)

然后在webpack.config.js文件中的plugins属性中配置

总结：在webpack.config.js文件中，一般module里的东西不用安装，直接设置。而plugins里的东西需要先安装再设置

以上的所有webpack.config.js文件配置结果如下：

const path = require('path');

const htmlPlugin= require('html-webpack-plugin');

module.exports={

//入口文件的配置项

entry:{

entry:'./src/entry.js'

},

//出口文件的配置项

output:{

//输出的路径，用了Node语法

path:path.resolve(\_\_dirname,'dist'),

//输出的文件名称

filename:'bundle.js'

},

//模块：例如解读CSS,图片如何转换，压缩

module:{},

//插件，用于生产模版和各项功能

plugins:[

new htmlPlugin({

minify:{

removeAttributeQuotes:true

},

hash:true,

template:'./src/index.html'

})

],

//配置webpack开发服务功能

devServer:{

//设置基本目录结构

contentBase:path.resolve(\_\_dirname,'dist'),

//服务器的IP地址，可以使用IP也可以使用localhost

host:'localhost',

//服务端压缩是否开启

compress:true,

//配置服务端口号

port:1717

}

}

**第四步：基本测试**

在package.json的scripts属性中设置如下

"scripts": {

"server": "webpack-dev-server",

"build": "webpack"

},

之后可以使用命令行npm run build来打包,然后npm run server来启动一个开发测试服务

**第五步：CSS打包**

本节是将文件关联的CSS都打包到出口的js文件中

5.1 CSS打包需要安装style-loader和css-loader

npm install --save-dev style-loader

npm install --save-dev css-loader

5.2 配置webpack.config.js

module:{

rules: [

{

test: /\.css$/,

use: [ 'style-loader', 'css-loader' ]

}

]

},

5.3 使用实例

例如创建一个CSS文件src/css/index.css并随便写点样式

然后在js文件中引入，例如在src/entery.js中引入inport css from ‘./css/index.css’;

然后运行打包npm run build

然后运行服务 npm run server 查看样式是否显示正确

**第六步：分离CSS**

工作中有时需要打包后的文件CSS是分离出来的

6.1 需要先安装npm install --save-dev extract-text-webpack-plugin

6.2 然后在webpack.config.js中引入

const extractTextPlugin = require("extract-text-webpack-plugin");

6.3 然后需要修改webpack.config.js文件中的module

module:{

rules: [

{

test: /\.css$/,

use: extractTextPlugin.extract({

fallback: "style-loader",

use: "css-loader"

})

}

]

},

6.4然后需要在webpack.config.js的plugins属性中增加一个对象

new extractTextPlugin("css/index.css")

6.4 测试打包后会在dist目录生成一个css文件夹

到目前，完整的webpack.config.js代码如下：

const path = require('path');

const htmlPlugin= require('html-webpack-plugin');

const extractTextPlugin = require("extract-text-webpack-plugin");

module.exports={

//入口文件的配置项

entry:{

entry:'./src/entry.js'

},

//出口文件的配置项

output:{

//输出的路径，用了Node语法

path:path.resolve(\_\_dirname,'dist'),

//输出的文件名称

filename:'bundle.js'

},

//模块：例如解读CSS,图片如何转换，压缩

module:{

rules: [

{

test: /\.css$/,

use: extractTextPlugin.extract({

fallback: "style-loader",

use: "css-loader"

})

}

]

},

//插件，用于生产模版和各项功能

plugins:[

new htmlPlugin({

minify:{

removeAttributeQuotes:true

},

hash:true,

template:'./src/index.html'

}),

new extractTextPlugin("css/index.css")

],

//配置webpack开发服务功能

devServer:{

//设置基本目录结构

contentBase:path.resolve(\_\_dirname,'dist'),

//服务器的IP地址，可以使用IP也可以使用localhost

host:'localhost',

//服务端压缩是否开启

compress:true,

//配置服务端口号

port:1717

}

}

**第二章：图片处理**

**第一步：CSS图片**

* 1. 需要先安装file-loader和url-loader

npm install --save-dev file-loader url-loader

* 1. 配置webpack.config.js文件的module，增加一个对象

1. {
2. test:/\.(png|jpg|gif|JPEG)/ ,
3. use:[{
4. loader:'url-loader',
5. options:{
6. limit:500, //如果这里数字很大，图片会已64位码的形式写到CSS文件里
7. outputPath:'images/' //使打包后的图片保存在dist/images中
8. }
9. }]
10. }

注意，limit的值如果写的很大，图片会已64位形式写在CSS里，改成500，打包后图片会单独分出来。

1.3 在webpack.config.js上方声明一个对象

var website ={

publicPath:"http://localhost:1717/"

}

然后在output选项中增加

publicPath:website.publicPath

1.4 测试

建立图片保存在src/images/里

然后在index.html中增加一个div id为tupian

然后在css里设置样式

#tupian{

   background-image: url(../images/manhua.png);

   width:466px;

   height:453px;

}

然后打包测试

目前为止：webpack.config.js文件内容如下：

const path = require('path');

const htmlPlugin= require('html-webpack-plugin');

const extractTextPlugin = require("extract-text-webpack-plugin");

var website ={

publicPath:"http://localhost:1717/"

}

module.exports={

//入口文件的配置项

entry:{

entry:'./src/entry.js'

},

//出口文件的配置项

output:{

//输出的路径，用了Node语法

path:path.resolve(\_\_dirname,'dist'),

//输出的文件名称

filename:'[name].js',

publicPath:website.publicPath

},

//模块：例如解读CSS,图片如何转换，压缩

module:{

rules: [

{

test: /\.css$/,

use: extractTextPlugin.extract({

fallback: "style-loader",

use: "css-loader"

})

},{

test:/\.(png|jpg|gif|JPEG)/ ,

use:[{

loader:'url-loader',

options:{

limit:500, //如果这里数字很大，图片会已64位码的形式写到CSS文件里

outputPath:'images/' //使打包后的图片保存在dist/images中

}

}]

}

]

},

//插件，用于生产模版和各项功能

plugins:[

new htmlPlugin({

minify:{

removeAttributeQuotes:true

},

hash:true,

template:'./src/index.html'

}),

new extractTextPlugin("css/index.css")

],

//配置webpack开发服务功能

devServer:{

//设置基本目录结构

contentBase:path.resolve(\_\_dirname,'dist'),

//服务器的IP地址，可以使用IP也可以使用localhost

host:'localhost',

//服务端压缩是否开启

compress:true,

//配置服务端口号

port:1717

}

}

**第二步：使用img标签**

2.1 需要先安装html-withimg-loader

Npm install html-withimg-loader --save-dev

2.2 配置webpack.config.js中的module选项，增加以下对象

{

test: /\.(htm|html)$/i,

use:[ 'html-withimg-loader']

}

2.3 测试

在index.html中插入一个img标签，然后测试图片是否显示正常

目前为止，webpack.config.js文件内容如下：

const path = require('path');

const htmlPlugin= require('html-webpack-plugin');

const extractTextPlugin = require("extract-text-webpack-plugin");

var website ={

publicPath:"http://localhost:1717/"

}

module.exports={

//入口文件的配置项

entry:{

entry:'./src/entry.js'

},

//出口文件的配置项

output:{

//输出的路径，用了Node语法

path:path.resolve(\_\_dirname,'dist'),

//输出的文件名称

filename:'[name].js',

publicPath:website.publicPath

},

//模块：例如解读CSS,图片如何转换，压缩

module:{

rules: [

{

test: /\.css$/,

use: extractTextPlugin.extract({

fallback: "style-loader",

use: "css-loader"

})

},{

test:/\.(png|jpg|gif|JPEG)/ ,

use:[{

loader:'url-loader',

options:{

limit:500, //如果这里数字很大，图片会已64位码的形式写到CSS文件里

outputPath:'images/' //使打包后的图片保存在dist/images中

}

}]

},{

test: /\.(htm|html)$/i,

use:[ 'html-withimg-loader']

}

]

},

//插件，用于生产模版和各项功能

plugins:[

new htmlPlugin({

minify:{

removeAttributeQuotes:true

},

hash:true,

template:'./src/index.html'

}),

new extractTextPlugin("css/index.css")

],

//配置webpack开发服务功能

devServer:{

//设置基本目录结构

contentBase:path.resolve(\_\_dirname,'dist'),

//服务器的IP地址，可以使用IP也可以使用localhost

host:'localhost',

//服务端压缩是否开启

compress:true,

//配置服务端口号

port:1717

}

}