命令 webpack 打包开发版本

webpack -p 打包发布版本

webpack中require-ensure和require-amd的区别

1. require-amd：同amd规范的require，依赖的模块下载并执行后，若有回调函数则调用回调函数。语法：

require(dependencies:string[],[callback:function(){}])

会生成chunkFile，有[id]无[name]，name用id

1. require-ensure：依赖的模块下载不执行，在回调函数中执行。语法：

require.ensure(dependencies：string[],callback:function,

[chunkFilename:string])

会生成chunkFile，有[id]。若存在chunkFilename则name是chunkFilename，否则name用id。若chunkFilename文件已经存在则本次依赖的模块合并到已经存在的文件中

webpack.config.js 指导通过webpack进行打包

webpack.config.js文件中：

（1）output中path是打包文件存放地，publicPath是部署完服务器后访问资源前加的公共路径。filename对应entry中出现的文件名，chunkFilename对应entry中未出现的文件名（如使用require-amd和require-ensure）

（2）entry 当值为一个时，[name]默认为main，当值为一个对象时[name]对应对象中每个属性名

（3）babel-loader：处理jsx文件

module:{

loaders:[

{test:/\.jsx$/,

exclude:/node\_modules/,

loader:”babel-loader?presets[]=es2015&presets[]=react”

}

]

}

test是匹配文件。exclude是不包括目录。loader是使用的加载方式及参数，其等于loader:”babel-loader”,

query:{

presets:[”es2015”,”react”]

}

babel-loader需要接收babel-preset-es2015插件读ES6

接收babel-preset-react插件读react

1. css-loader：处理css文件

module：{

loaders:[{

test:/\.css$/,

loader:”style-loader!css-loader”

}]

}

先用css-loader读css文件，然后用style-loader将style插入到html中。css-loader默认为全局css，当传入modules参数后为模块css，若想将某个css转成全局css，用:global(选择器){样式}。

1. image-loader：处理图片文件

module：{

loaders:[{

test:/\.(png|jpg)$/,

loader:”url-loader?limit=8192”

}]

}

如果图片比8192 bytes小，将被转成Data URL形式。

如果图片比8192 bytes大，将被转成正常 URL形式。

webpack有插件系统可扩展功能：

UglifyJsPlugin：将打包后的js文件缩小。webpack.config.js文件如下：

var webpack = require(”webpack”);

var uglifyJsPlugin = webpack.optimize.UglifyJsPlugin;

module.exports={

…

plugins:[

new uglifyJsPlugin({

compress:{

warnings：false

}

})

]

}

html-webpack-plugin和open-browser-webpack-plugin等第三方插件。html-webpack-plugin帮助创建index.html。open-browser-webpack-plugin

帮助打开一个新的浏览器tab。

webpack.config.js如下

var HtmlwebpackPlugin = require(”html-webpack-plugin”);

var OpenBrowserPlugin= require(”open-browser-webpack-plugin”);

module.export= {

…

plugins:[

new HtmlwebpackPlugin ({

title:”webpack-demos”,

filename：”index.html”

}),

new OpenBrowserPlugin({

url:”http://localhost:8080”

}),

]

};

commonsChunkPlugin提取chunkFile公共部分到单独文件

webpack.config.js如下

var CommonsChunkPlugin=

require(”webpack/lib/optimize/CommonChunkPlugin”);

module.export= {

…

plugins:[

new CommonsChunkPlugin(”init.js”)

]

}

单独文件命名为init.js

commonsChunkPlugin可以打包jquery等类库到单独文件

main.js如下

var $=require(”jquery”);

$(”h1”).text(”Hello World”);

index.html如下

<html>

<body>

<h1></h1>

<script src=”vendor.js”></script>

<script src=”bundle.js”></script>

</body>

</html>

webpack.config.js如下

var webpack= require(”webpack”);

module.exports= {

entry:{

app:”./main.js”,

vendor:[”jquery”]

}

output:{

filename:”bundle.js”

}

plugins:[

new webpack.optimize.CommonsChunkPlugin(”vendor”,

”vendor.js”)

]

}

ProvidePlugin 当所有模块都需要一个模块如jquery，可不用每个模块都require，用ProvidePlugin即可

main.js如下

$(”h1”).text(”Hello World”);

webpack.config.js如下

var webpack= require(”webpack”);

module.exports = {

entry:{

app:”./main.js”

},

output:{

filename:”bundle.js”

},

plugins:[

new webpack.ProvidePlugin({

$:”jquery”,

jQuery:”jquery”,

”window.jQuery”:”jquery”

})

]

}

externals如果想使用某个文件，而不将其打包，可以：

data.js如下

var data = ”Hello World”;

webpack.config.js如下

module.exports = {

entry:app:”./main.js”,

…

externals:{

”data”:”data”

}

}

main.js如下

var data= require(”data”);

…

热模快替换

1. 命令行式

webpack-dev-server --hot --inline

--hot 添加热模快插件，并将服务器转成热模式

--inline 将webpack-dev-server/client?http://localhost:8080打包进去

--hot --inline 以上和将webpack/hot/dev-server打包进去

（2）webpack.config.js如下

var webpack= require(”webpack”);

module.exports = {

entry:[

”webpack/hot/dev-server”,

”webpack-dev-server/client?http://localhost:8080”,

”./index.js”

],

…

plugins:[

new webpack.HotModuleReplacementPlugin()

],

…

}

命令行为webpack-dev-server