

Webpack-Day2



- loader: file-loader: 处理静态资源模块

loader: file-loader

原理是把打包入口中识别出的资源模块，移动到输出目录，并且返回一个地址名称

所以我们什么时候用file-loader呢？

场景：就是当我们需要模块，仅仅是从源代码挪移到打包目录，就可以使用file-loader来处理，txt, svg, csv, excel, 图片资源啦等等

```
npm install file-loader -D
```

案例：

```
module: {
  rules: [
    {
      test: /\.?(png|jpe?g|gif)$/ ,
      //use使用一个loader可以用对象，字符串，两个loader需要用数组
      use: {
        loader: "file-loader",
        // options额外的配置，比如资源名称
        options: {
          // placeholder 占位符 [name]老资源模块的名称
          // [ext]老资源模块的后缀
          // https://webpack.js.org/loaders/file-loader#placeholders
          name: "[name]_[hash].[ext]",

```

开课吧web全栈架构师

```

        //打包后的存放位置
        outputPath: "images/"
    }
}
}
]
},

```

```

import pic from "./logo.png";

var img = new Image();
img.src = pic;
img.classList.add("logo");

var root = document.getElementById("root");
root.append(img);

```

- 处理字体 <https://www.iconfont.cn/?spm=a313x.7781069.1998910419.d4d0a486a>

```

//css
@font-face {
  font-family: "webfont";
  font-display: swap;
  src: url("webfont.woff2") format("woff2");
}

body {
  background: blue;
  font-family: "webfont" !important;
}

//webpack.config.js
{
  test: /\. (eot|ttf|woff|woff2|svg)$/,
  use: "file-loader"
}

```

- url-loader file-loader加强版本

url-loader内部使用了file-loader,所以可以处理file-loader所有的事情,但是遇到jpg格式的模块,会把该图片转换成base64格式字符串,并打包到js里。对小体积的图片比较合适,大图片不合适。

```
npm install url-loader -D
```

案例;

```
module: {
  rules: [
    {
      test: /\. (png|jpe?g|gif)$/ ,
      use: {
        loader: "url-loader",
        options: {
          name: "[name]_[hash].[ext]",
          outputPath: "images/",
          //小于2048, 才转换成base64
          limit: 2048
        }
      }
    }
  ]
},
```

样式处理:

Css-loader 分析css模块之间的关系,并合成一个css

Style-loader 会把css-loader生成的内容,以style挂载到页面的heade部分

```
npm install style-loader css-loader -D
```

```
{
  test: /\.css$/,
  use: ["style-loader", "css-loader"]
}

{
```

```

test: /\.css$/,
use: [{
  loader: "style-loader",
  options: {
    injectType: "singletonStyleTag" // 把所有的style标签合并成一个
  }
}, "css-loader"]
}

```

Less样式处理

less-load 把less语法转换成css

```
$ npm install less less-loader --save-dev
```

案例：

loader有顺序，从右到左，从下到上

```

{
  test: /\.scss$/,
  use: ["style-loader", "css-loader", "less-loader"]
}

```

样式自动添加前缀：

<https://caniuse.com/>

Postcss-loader

```
npm i postcss-loader autoprefixer -D
```

新建postcss.config.js

```

//webpack.config.js
{
  test: /\.css$/,
  use: ["style-loader", "css-loader", "postcss-loader"]
},

//postcss.config.js
module.exports = {
  plugins: [

```

```
require("autoprefixer")({
  overrideBrowserslist: ["last 2 versions", ">1%"]
})
]
};
```

loader 处理webpack不支持的格式文件，模块

一个loader只处理一件事情

loader有执行顺序

如何自己编写一个Loader

自己编写一个Loader的过程是比较简单的，

Loader就是一个函数，**声明式函数**，不能用箭头函数

拿到源代码，作进一步的修饰处理，再返回处理后的源码就可以了

官方文档：<https://webpack.js.org/contribute/writing-a-loader/>

接口文档：<https://webpack.js.org/api/loaders/>

简单案例

- 创建一个替换源码中字符串的loader

```
//index.js
console.log("hello kkb");
```

```
//replaceLoader.js
module.exports = function(source) {
  console.log(source, this, this.query);
  return source.replace('kkb', '开课吧')
};
```

//需要用声明式函数，因为要上到上下文的this,用到this的数据，该函数接受一个参数，是源码

- 在配置文件中使用时loader

```
//需要使用node核心模块path来处理路径
const path = require('path')
module: {
  rules: [
    {
      test: /\.js$/,
      use: path.resolve(__dirname, './loader/replaceLoader.js')
    }
  ]
},
```

- 如何给loader配置参数，loader如何接受参数?

- this.query
- loader-utils

```
//webpack.config.js
module: {
  rules: [
    {
      test: /\.js$/,
      use: [
        {
          loader: path.resolve(__dirname, './loader/replaceLoader.js'),
          options: {
            name: "开课吧"
          }
        }
      ]
    }
  ]
},

//replaceLoader.js
//const loaderUtils = require("loader-utils");//官方推荐处理loader,query的工具

module.exports = function(source) {
  //this.query 通过this.query来接受配置文件传递进来的参数

  //return source.replace("kkb", this.query.name);
  const options = loaderUtils.getOptions(this);
  const result = source.replace("kkb", options.name);
  return source.replace("kkb", options.name);
}
```

- **this.callback**: 如何返回多个信息, 不止是处理好的源码呢, 可以使用this.callback来处理

```
//replaceLoader.js
const loaderUtils = require("loader-utils");//官方推荐处理loader,query的工具

module.exports = function(source) {
  const options = loaderUtils.getOptions(this);
  const result = source.replace("kkb", options.name);
  this.callback(null, result);
};

//this.callback(
  err: Error | null,
  content: string | Buffer,
  sourceMap?: SourceMap,
  meta?: any
);
```

- **this.async**: 如果loader里面有异步的事情要怎么处理呢

```
const loaderUtils = require("loader-utils");

module.exports = function(source) {
  const options = loaderUtils.getOptions(this);
  setTimeout(() => {
    const result = source.replace("kkb", options.name);

    return result;
  }, 1000);
};

//先用setTimeout处理下试试, 发现会报错
```

我们使用this.async来处理, 他会返回this.callback

```

const loaderUtils = require("loader-utils");

module.exports = function(source) {
  const options = loaderUtils.getOptions(this);

  //定义一个异步处理, 告诉webpack, 这个loader里有异步事件, 在里面调用下这个异步
  //callback 就是 this.callback 注意参数的使用
  const callback = this.async();
  setTimeout(() => {
    const result = source.replace("kkb", options.name);
    callback(null, result);
  }, 3000);
};

```

- 多个loader的使用

```

//replaceLoader.js
module.exports = function(source) {
  return source.replace("开课吧", "word");
};

//replaceLoaderAsync.js
const loaderUtils = require("loader-utils");
module.exports = function(source) {
  const options = loaderUtils.getOptions(this);
  //定义一个异步处理, 告诉webpack, 这个loader里有异步事件, 在里面调用下这个异步
  const callback = this.async();
  setTimeout(() => {
    const result = source.replace("kkb", options.name);
    callback(null, result);
  }, 3000);
};

//webpack.config.js
module: {
  rules: [
    {
      test: /\.js$/,
      use: [
        path.resolve(__dirname, "../loader/replaceLoader.js"),
        {
          loader: path.resolve(__dirname, "../loader/replaceLoaderAsync.js"),
          options: {
            name: "开课吧"
          }
        }
      ]
    }
  ]
}
// use: [path.resolve(__dirname, "../loader/replaceLoader.js")]

```



```
    }  
  ]  
},
```

顺序，自下而上，自右到左

- 处理loader的路径问题

```
resolveLoader: {  
  modules: ["node_modules", "./loader"]  
},  
module: {  
  rules: [  
    {  
      test: /\.js$/,  
      use: [  
        "replaceLoader",  
        {  
          loader: "replaceLoaderAsync",  
          options: {  
            name: "开课吧"  
          }  
        }  
      ]  
      // use: [path.resolve(__dirname, "./loader/replaceLoader.js")]  
    }  
  ]  
},  
},
```

参考：loader API

<https://webpack.js.org/api/loaders>

mini-css-extract-plugin

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin");

{
  test: /\.css$/,
  use: [MiniCssExtractPlugin.loader, "css-loader"]
}

new MiniCssExtractPlugin({
  filename: "[name][chunkhash:8].css"
})
```

chunk: 一个入口肯定是一个chunk, 但是一个chunk不一定只有一个依赖

hash

Chunkhash

Contenthash