Webpack-内容补充

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.asycn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 注意顺序

```
//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

如何自己编写一个Plugins

Plugin: 开始打包，在某个时刻，帮助我们处理一些什么事情的机制

plugin要比loader稍微复杂一些，在webpack的源码中，用plugin的机制还是占有非常大的场景，可以说plugin是webpack的灵魂

设计模式

事件驱动

发布订阅

plugin是一个类，里面包含一个apply函数，接受一个参数，compiler

官方文档：https://webpack.js.org/contribute/writing-a-plugin/

案例：

- 创建copyright-webpack-plugin.js

```
class CopyrightWebpackPlugin {
  constructor() {
  }

  //compiler：webpack实例
  apply(compiler) {

  }
}
module.exports = CopyrightWebpackPlugin;
```

- 配置文件里使用

```
const CopyrightWebpackPlugin = require("./plugin/copyright-webpack-plugin");

plugins: [new CopyrightWebpackPlugin()]
```

- 如何传递参数

```
//webpack配置文件
plugins: [
    new CopyrightWebpackPlugin({
      name: "开课吧"
    })
  ]

//copyright-webpack-plugin.js
class CopyrightWebpackPlugin {
  constructor(options) {
    //接受参数
    console.log(options);
```

```
  }

  apply(compiler) {}
}
module.exports = CopyrightWebpackPlugin;
```

- 配置plugin在什么时刻进行

```
class CopyrightWebpackPlugin {
  constructor(options) {
    // console.log(options);
  }

  apply(compiler) {
    //hooks.emit 定义在某个时刻
    compiler.hooks.emit.tapAsync(
      "CopyrightWebpackPlugin",
      (compilation, cb) => {
        compilation.assets["copyright.txt"] = {
          source: function() {
            return "hello copy";
          },
          size: function() {
            return 20;
          }
        };
        cb();
      }
    );

    //同步的写法
    //compiler.hooks.compile.tap("CopyrightWebpackPlugin",
compilation => {
    //  console.log("开始了");
```

```
    //});
  }
}
module.exports = CopyrightWebpackPlugin;
```

参考：compiler-hooks

https://webpack.js.org/api/compiler-hooks

node调试工具使用

- 修改scripts

```
    "debug": "node --inspect --inspect-brk
node_modules/webpack/bin/webpack.js"
```

end