loader

作用

用于对模块的源代码进行转换

在使用import语句或者require()加载模块时预处理文件

loader可以将不同的语言(如TypeScript)转换为JavaScript或者将内联图像转换为data URL

甚至允许直接在JavaScript模块中import CSS文件

以加载CSS文件以及转为TypeScript为例

首先安装响应的loader：

npm install --save-dev css-loader ts-loader

然后设置规则，为所有css后缀的文件使用css-loader, 为所有.ts后缀文件使用ts-loader

```
module.exports = {
   module:{
      rules: [
         { test: /\.css$/, use: 'css-loader' },
         { test: /\.ts$/, use: 'ts-loader' },
      ]
   }
}
```

两种使用loader的方式

配置方式 / 在webpack.config.js文件中指定loader

可以在webpack配置中给同一类型文件指定多个loader，即use属性对应一个数组类型

在执行的时候按照数组从后往前执行，如下，顺序为

sass-loader -> css-loader -> style-loader

```
module.exports = {
   module:{
      rules: [
         {
            test: /\.css$/,
            use: [
               // style-loader
               { loader: "style-loader" },
               // css-loader
               {
                  loader: "css-loader",
                  options: {
                     modules: true
                  }
               },
               // sass-loader
               { loader: "sass-loader" }
            ]
         },
      ]
   }
}
```

内联方式 / 在每个import语句中显示指定loader

在import语句等引用方式中指定loader

使用!将资源中的loader分开

```
import Styles from 'style-loader!css-loader?modules!./styles.css';
```

添加前缀用于覆盖配置中的loader等，有三种前缀

- !前缀，禁用所有已配置的normal loader
- !!前缀，禁用所有已配置的loader (preLoader, loader,postLoader)
- -!前缀，禁用所有已配置的preLoader和loader，但是不禁用postLoaders

同时设置loader时可以传递参数，类似于URI中的search部分，以？开头，不同的键值对以&分开，同样也可以使用一个JSON对象传参，如

?{"key":"value","foo","bar"}

尽量使用配置方式module.rules，更方便调试以及定位

loader特性

如上将use配置为一个loader数组，并且每个loader使用对象语法，可以带上option参数

支持链式调用

- 链中的每个loader会链式传递以及处理过的资源
- 链式的loader转换的过程时按照属性的相反顺序执行

可以是同步的，也可以是异步的

运行在Node.js中，能够执行任何操作

每个loader都可以通过options对象配置

除了通过package.json中的main来将一个npm模块导出为loader

同样可以在module.rules中使用loader字段直接引用一个模块

插件能够为loader带来更多特性

能够产生额外的任意文件

可以通过loader的预处理函数，加入一些细粒度的逻辑

- 压缩
- 打包
- 语言转译 / 编译

解析loader

遵循标准模块解析规则

从模块路径加载

- npm install
- node_modules进行加载

loader模块将导出为一个函数，并且编写为Node.js兼容的JavaScript