在之前打包的时候，我们都是把代码在这个里面进行了相应的处理，然后直接生成就可以了，但是在真正的开发过程当中， 如果我们要去使用这个 `webpack` 是不能够这么做的，因为打包是需要花费时间的，我们如果有一个小小的改动就要去打包，这样做非常麻烦，所以我们希望得到一个点就是边开发，边看最终的预览效果，这一种模式我们叫webpack的开发环境

开发环境

> **Tip**
>
> 本指南继续沿用 管理输出 指南中的代码示例。

如果你一直跟随之前的指南，应该对一些 webpack 基础知识有着很扎实的理解。在我们继续之前，先来看看如何设置一个开发环境，使我们的开发体验变得更轻松一些。

> **Warning**
>
> 本指南中的工具**仅用于开发环境**，请**不要**在生产环境中使用它们！

在开始前，我们先将 `mode` 设置为 `'development'`，并将 `title` 设置为 `'Development'`。

> 最先最重要的一个点就是我们需要实现一个预览的功能，我只要改更了源代码，它立刻出现后面的效果，如果整个效果都没有问题了，那么我们最终才会生成 `dist` 目录里面的代码

要实现开发环境的配置，我们需要使用第三方的一个工具叫 `webpack-dev-server`

安装包

```
$ yarn add webpack-dev-server --dev
```

在package.json里面配置开发的启动命令

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack --config ./webpack.config.js",
    "dev": "webpack-dev-server --config ./webpack.config.dev.js"
},
```

编写配置文件

在当前的项目目录下面，新建一个 `webpack.config.dev.js` 这一个配置文件，这一个配置文件也是 `webpack` 的配置文件，它里面有80%的东西与之前的生产模式是一相同的

```
/**
 * 开发环境的配置
 */

const path = require("path");
const HTMLWebapckPlugin = require("html-webpack-plugin");
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
const webpack = require("webpack");
const CopyWebpackPlugin = require("copy-webpack-plugin");

/**
```

```
 * @type {import("webpack").Configuration}
 */
const config = {
    target: ["web", "es5"],
    mode: "development",
    //入口,多页面的入口 ，可以以对象的形式去完成
    entry: {
        index: "./js/index.js",
        login: "./js/login.js"
    },
    output: {
        path: path.join(__dirname, "./dist"),
        filename: "js/[name].[fullhash:8].js",
    },
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/,
                // use:["babel-loader"]
                use: [
                    {
                        loader: "babel-loader"
                    }
                ]
            }, {
                test: /\.css$/,
                use: [
                    "style-loader",
                    {
                        loader: "css-loader",
                        options: {
                            importLoaders: 1
                        }
                    },
                    "postcss-loader"
                ]
            },
            {
                test: /\.s[ca]ss$/,
                use: [
                    "style-loader",
                    {
                        loader: "css-loader",
                        options: {
                            importLoaders: 2
                        }
                    },
                    "postcss-loader",
                    "sass-loader"
                ]
            },

            //在开发模式下面，文件就不用管了
        ]
    },
```

```
    plugins: [
        new webpack.ProgressPlugin(),
        //定义全局变量，如jquery
        new webpack.ProvidePlugin({
            "$": "jquery",
            "jQuery": "jquery",
            "window.jquery": "jquery"
        }),
        new HTMLWebapckPlugin({
            filename: "index.html",
            template: path.join(__dirname, "./index.html"),
            inject: true,
            chunks: ["index"]
        }),
        new HTMLWebapckPlugin({
            filename: "login.html",
            template: path.join(__dirname, "./login.html"),
            inject: true,
            chunks: ["login"]
        }),
        new CopyWebpackPlugin({
            patterns: [
                {
                    from: path.join(__dirname, "./static"),
                    to: path.join(__dirname, "./dist/static")
                }
            ]
        }),
    ],
    devServer: {
        host: "0.0.0.0",
        port: 8989,
        allowedHosts: "*",
        static: {
            directory: path.join(__dirname, "./dist/static")
        },
        //如果报错了，直接在客户端显示出来
        client: {
            overlay: true
        },
        //在没有配置这个`watchFiles`的情况下，如果我们更改html里面的内容 ，它是不会自动刷新的
        watchFiles: [
            "./index.html",
            "./login.html"
        ]
    }
}
module.exports = config;
```