link: null
title: 珠峰架构师成长计划
description: package.json
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=125 sentences=293, words=1457

1. 什么是HMR

- Hot Module Replacement是指当你对代码修改并保存后，webpack将会对代码进行得新打包，并将新的模块发送到浏览器端，浏览器用新的模块替换掉旧的模块，以实现在不刷新浏览器的前提下更新页面。
- 相对于 live reload刷新页面的方案，HMR的优点在于可以保存应用的状态,提高了开发效率

2. 搭建HMR项目

2.1 安装依赖的模块

```
cnpm i webpack@4.39.1 webpack-cli@3.3.6 webpack-dev-server@3.7.2 mime html-webpack-plugin express socket.io -S
```

2.2 package.json

package.json

```
{
  "name": "zhufeng_hmr",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "webpack",
    "dev": "webpack-dev-server"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "webpack": "4.39.1",
    "webpack-cli": "3.3.6",
    "webpack-dev-server": "3.7.2"
  }
}
```

2.2 webpack.config.js

webpack.config.js

```
const path = require('path');
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    mode:'development',
    entry: './src/index.js',
    output: {
        filename: 'main.js',
        path: path.join(__dirname, 'dist')
    },
    devServer: {
        contentBase:path.join(__dirname, 'dist')
    },
    plugins:[
        new HtmlWebpackPlugin({
            template:'./src/index.html',
            filename:'index.html'
        })
    ]
}
```

2.3 src\index.js

src\index.js

```
let root = document.getElementById('root');
function render(){
   let title = require('./title').default;
   root.innerHTML= title;
}
render();
```

2.4 src\title.js

src\title.js

```
export default 'hello';
```

2.5 src\index.html

src\index.html

```
   webpack热更新
```

2.6 dist\bundle.js

dist\main.js

```
(function(modules) {
    var installedModules = {};

    function __webpack_require__(moduleId) {

        if(installedModules[moduleId]) {

            return installedModules[moduleId].exports;
        }

        var module = installedModules[moduleId] = {
            i: moduleId,
            l: false,
            exports: {}
        };

        modules[moduleId].call(module.exports, module, module.exports, __webpack_require__);

        module.l = true;

        return module.exports;
    }

    __webpack_require__.m = modules;

    __webpack_require__.c = installedModules;

    __webpack_require__.d = function(exports, name, getter) {

        if(!__webpack_require__.o(exports, name)) {

            Object.defineProperty(exports, name, { enumerable: true, get: getter });
        }
    };

    __webpack_require__.r = function(exports) {

        if(typeof Symbol !== 'undefined' && Symbol.toStringTag) {
            Object.defineProperty(exports, Symbol.toStringTag, { value: 'Module' });
        }
        Object.defineProperty(exports, '__esModule', { value: true });
    };

    __webpack_require__.t = function(value, mode) {
        if(mode & 1) value = __webpack_require__(value);
        if(mode & 8) return value;

        if((mode & 4) && typeof value === 'object' && value && value.__esModule) return value;

    var ns = Object.create(null);
        __webpack_require__.r(ns);
        Object.defineProperty(ns, 'default', { enumerable: true, value: value });

        if(mode & 2 && typeof value != 'string') for(var key in value) __webpack_require__.d(ns, key, function(key) { return value[key]; }.bind(null, key));

    return ns;
    };

    __webpack_require__.n = function(module) {

        var getter = module && module.__esModule ?
            function getDefault() { return module['default']; } :
            function getModuleExports() { return module; };

        __webpack_require__.d(getter, 'a', getter);
        return getter;
    };

    __webpack_require__.o = function(object, property) { return Object.prototype.hasOwnProperty.call(object, property); };

    __webpack_require__.p = "";

    return __webpack_require__(__webpack_require__.s = "./src/index.js");
})
({

 "./src/index.js":
 (function(module, __webpack_exports__, __webpack_require__) {
eval(`
__webpack_require__.r(__webpack_exports__);//因为是es模块，所以要添加__esModule属性
var _title__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(\"./src/title.js\");
function render(){
  let root = document.getElementById('root');
  root.innerHTML= _title__WEBPACK_IMPORTED_MODULE_0__[\"default\"];
}
render();`);
}),

 "./src/title.js":
 (function(module, __webpack_exports__, __webpack_require__) {
eval(`
__webpack_require__.r(__webpack_exports__);//因为是es模块，所以要添加__esModule属性
__webpack_exports__[\"default\"] = ('hello');
`);
 })
});
```

3. webpack的编译流程

- 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数； 开始编译：用上一步得到的参数初始化 `Compiler` 对象，加载所有配置的插件，执行对象的 `run` 方法开始执行编译；
- 确定入口：根据配置中的 entry 找出所有的入口文件；
- 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行翻译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理；
- 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系；

- 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会；
- 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统。

> 在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用 Webpack 提供的 API 改变 Webpack 的运行结果。 chunk 就是若干 module 打成的包，一个 chunk 应该包括多个 module，一般来说最终会形成一个 file。而 js 以外的资源，webpack 会通过各种 loader 转化成一个 module，这个模块会被打包到某个 chunk 中，并不会形成一个单独的 chunk

3. 实现热更新

3.1 webpack.config.js

webpack.config.js

```
module.exports = {
    devServer:{
+       hot:true,
        contentBase:path.join(__dirname,'dist')
    },
    plugins:[
+       new webpack.HotModuleReplacementPlugin()
    ]
}
```

3.2 index.js

src\index.js

```
import './client';
let root = document.getElementById('root');
function render(){
   let title = require('./title').default;
   root.innerHTML= title;
}
render();

+if(module.hot){
+  module.hot.accept(['./title'],()=>{
+      render();
+  });
+}
```

4. debug

debugger.js

```
debugger
require('./node_modules/webpack-dev-server/bin/webpack-dev-server.js');
```

5. 源代码位置

5.1. 服务器部分

1. 启动webpack-dev-server服务器
2. 创建webpack实例
3. 创建Server服务器
4. 添加webpack的 done事件回调，在编译完成后会向浏览器发送消息
5. 创建express应用app
6. 使用监控模式开始启动webpack编译,在 webpack 的 watch 模式下，文件系统中某一个文件发生修改，webpack 监听到文件变化，根据配置文件对模块重新编译打包，并将打包后的代码通过简单的 JavaScript 对象保存在内存中
7. 设置文件系统为内存文件系统
8. 添加webpack-dev-middleware中间件
9. 创建http服务器并启动服务
10. 使用sockjs在浏览器端和服务端之间建立一个 websocket 长连接，将 webpack 编译打包的各个阶段的状态信息告知浏览器端,浏览器端根据这些 socket消息进行不同的操作。当然服务端传递的最主要信息还是新模块的 hash值，后面的步骤根据这一 hash值来进行模块热替换

步骤 代码位置 1.启动webpack-dev-server服务器
webpack-dev-server.js#L159 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/bin/webpack-dev-server.js#L73)

2.创建webpack实例
webpack-dev-server.js#L89 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/bin/webpack-dev-server.js#L89)

3.创建Server服务器
webpack-dev-server.js#L100 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/bin/webpack-dev-server.js#L107)

4. 添加webpack的 done

事件回调
Server.js#L120 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L122)

编译完成向客户端发送消息
Server.js#L183 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L184)

5.创建express应用app
Server.js#L121 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L123)

6. 添加webpack-dev-middleware中间件
   Server.js#L121 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L121)

中间件负责返回生成的文件
middleware.js#L20 (https://github.com/webpack/webpack-dev-middleware/blob/v3.7.0/lib/middleware.js#L20)

启动webpack编译
index.js#L51 (https://github.com/webpack/webpack-dev-middleware/blob/v3.7.0/index.js#L51)

7. 设置文件系统为内存文件系统
   fs.js#L115 (https://github.com/webpack/webpack-dev-middleware/blob/v3.7.0/lib/fs.js#L115)

8. 创建http服务器并启动服务
   Server.js#L135 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L135)

9. 使用sockjs在浏览器端和服务端之间建立一个 websocket 长连接
   Server.js#L745 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js#L745)

创建socket服务器
SockJSServer.js#L34 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/servers/SockJSServer.js#L34)

5.2. 客户端部分

1. webpack-dev-server/client-src/default/index.js端会监听到此 hash消息,会保存此hash值
2. 客户端收到 ok的消息后会执行 reloadApp方法进行更新
3. 在reloadApp中会进行判断，是否支持热更新，如果支持的话发射 webpackHotUpdate事件,如果不支持则直接刷新浏览器
4. 在 webpack/hot/dev-server.js会监听 webpackHotUpdate事件,然后执行 check()方法进行检查
5. 在check方法里会调用 module.hot.check方法
6. 它通过调用 JsonpMainTemplate.runtime的 hotDownloadManifest方法，向 server 端发送 Ajax 请求，服务端返回一个 Manifest文件，该 Manifest 包含了所有要更新的模块的 hash 值和chunk名
7. 调用 JsonpMainTemplate.runtime的 hotDownloadUpdateChunk方法通过JSONP请求获取到最新的模块代码
8. 补丁JS取回来后会调用 JsonpMainTemplate.runtime.js的 webpackHotUpdate方法，里面会调用 hotAddUpdateChunk方法,用新的模块替换掉旧的模块
9. 然后会调用 HotModuleReplacement.runtime.js的 hotAddUpdateChunk方法动态更新模块代 码
10. 然后调用 hotApply方法进行热更新

步骤 代码 1. webpack-dev-server/client

端会监听到此hash消息
index.js#L54 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/client-src/default/index.js#L54)

2. 客户端收到 ok

的消息后会执行 reloadApp

方法进行更新
index.js#L101 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/client-src/default/index.js#L101)

3. 在reloadApp中会进行判断，是否支持热更新，如果支持的话发射 webpackHotUpdate

事件,如果不支持则直接刷新浏览器
reloadApp.js#L7 (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/client-src/default/utils/reloadApp.js#L7)

4. 在 webpack/hot/dev-server.js

会监听 webpackHotUpdate

事件
dev-server.js#L55 (https://github.com/webpack/webpack/blob/v4.39.1/hot/dev-server.js#L55)

5. 在check方法里会调用 module.hot.check

方法
dev-server.js#L13 (https://github.com/webpack/webpack/blob/v4.39.1/hot/dev-server.js#L13)

6. HotModuleReplacement.runtime

请求Manifest
HotModuleReplacement.runtime.js#L180 (https://github.com/webpack/webpack/blob/v4.39.1/lib/HotModuleReplacement.runtime.js#L180)

7. 它通过调用 JsonpMainTemplate.runtime hotDownloadManifest

方法
JsonpMainTemplate.runtime.js#L23 (https://github.com/webpack/webpack/blob/v4.39.1/lib/web/JsonpMainTemplate.runtime.js#L23)

8. 调用 JsonpMainTemplate.runtime hotDownloadUpdateChunk

方法通过JSONP请求获取到最新的模块代码
JsonpMainTemplate.runtime.js#L14 (https://github.com/webpack/webpack/blob/v4.39.1/lib/web/JsonpMainTemplate.runtime.js#L14)

9. 补丁JS取回来后会调用 JsonpMainTemplate.runtime.js webpackHotUpdate

方法
JsonpMainTemplate.runtime.js#L8 (https://github.com/webpack/webpack/blob/v4.39.1/lib/web/JsonpMainTemplate.runtime.js#L8)

10. 然后会调用 HotModuleReplacement.runtime.js hotAddUpdateChunk

方法动态更新模块代码
HotModuleReplacement.runtime.js#L222 (https://github.com/webpack/webpack/blob/v4.39.1/lib/HotModuleReplacement.runtime.js#L222)

11.然后调用 hotApply

方法进行热更新
HotModuleReplacement.runtime.js#L257 (https://github.com/webpack/webpack/blob/v4.39.1/lib/HotModuleReplacement.runtime.js#L257) HotModuleReplacement.runtime.js#L278 (https://github.com/webpack/webpack/blob/v4.39.1/lib/HotModuleReplacement.runtime.js#L278)

5.3 相关代码

- webpack-dev-server.js (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/bin/webpack-dev-server.js)
- Server.js (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/Server.js)
- webpack-dev-middleware/index.js (https://github.com/webpack/webpack-dev-middleware/blob/v3.7.0/index.js)
- SockJSServer.js (https://github.com/webpack/webpack-dev-server/blob/v3.7.2/lib/servers/SockJSServer.js)

6. 实现热更新

6.1 webpack-dev-server.js

```
const path = require('path');
const express = require('express');
const mime = require('mime');
const webpack = require('webpack');

let config = require('./webpack.config');
let compiler = webpack(config);
class Server{
    constructor(compiler){
        this.compiler = compiler;

        let lastHash;
        let sockets=[];
        compiler.hooks.done.tap('webpack-dev-server', (stats) => {
           lastHash = stats.hash;
           sockets.forEach(socket=>{
             socket.emit('hash',stats.hash);
             socket.emit('ok');
           });
        });

        let app = new express();

        compiler.watch(config.watchOptions||{}, (err)=>{
            console.log('编译成功');
        });

        const MemoryFileSystem = require('memory-fs');
        const fs = new MemoryFileSystem();;
        compiler.outputFileSystem = fs;

        const devMiddleware = (req,res,next)=>{
            if(req.url === '/favicon.ico'){
                return res.sendStatus(404);
            }
            let filename = path.join(config.output.path,req.url.slice(1));
            console.error(filename);
            if(fs.statSync(filename).isFile()){
                let content = fs.readFileSync(filename);
                res.header('Content-Type',mime.getType(filename));
                res.send(content);
            }else{
                next();
            }
        }
        app.use(devMiddleware);

        this.server = require('http').createServer(app);

        let io = require('socket.io')(this.server);
        io.on('connection',(socket)=>{
            sockets.push(socket);
            if(lastHash){
                socket.emit('hash',lastHash);
                socket.emit('ok');
            }
        });
    }

    listen(port){
         this.server.listen(port,()=>{
            console.log(port+'服务启动成功!')
        });
    }
}

let server = new Server(compiler);
server.listen(8000);
```

6.2 client.js

src\client.js

```
let socket = io('/');
class Emitter{
  constructor(){
    this.listeners = {};
  }
  emit(type){
    this.listeners[type]&&this.listeners[type]();
  }
  on(type,listener){
    this.listeners[type] = listener;
  }
}
const hotEmitter= new Emitter();

let hotCurrentHash;
let currentHash;
const onConnected = ()=>{
  console.log('客户端已经连接');

  socket.on('hash',(hash)=>{
    currentHash = hash;
  });

  socket.on('ok',()=>{
      reloadApp(true);
  });
  socket.on('disconnect',()=>{
    hotCurrentHash=currentHash=null;
  });
}

function reloadApp(hot){
  if(!hot){
    return window.location.reload();
  }
  hotEmitter.emit('webpackHotUpdate');
}

hotEmitter.on("webpackHotUpdate", function() {
  if(!hotCurrentHash || hotCurrentHash === currentHash){
      return hotCurrentHash = currentHash;
  }

  hotCheck();
});
function hotCheck(){

  hotDownloadManifest().then(update=>{
      let chunkIds = Object.keys(update.c);
      chunkIds.forEach((chunkId)=>{

            hotDownloadUpdateChunk(chunkId);
      });
  });
}

function hotDownloadUpdateChunk(chunkId) {
  var script = document.createElement("script");
  script.charset = "utf-8";
  script.src = "/" + chunkId + "." + hotCurrentHash + ".hot-update.js";
  document.head.appendChild(script);
}
function hotDownloadManifest(){
  return new Promise((resolve,reject)=>{
   var request = new XMLHttpRequest();
   var requestPath = "/" + hotCurrentHash + ".hot-update.json";
   request.open("GET", requestPath, true);
   request.onreadystatechange = function() {
    if(request.readyState == 4){
     let update = JSON.parse(request.responseText);
     resolve(update);
    }
   }
   request.send();
  });
}

window.webpackHotUpdate = (chunkId, moreModules)=>{
  for(let moduleId in moreModules){
    let oldModule =  __webpack_require__.c[moduleId];
    let {parents,children} = oldModule;
    var module = __webpack_require__.c[moduleId] = {
          i: moduleId,
          l: false,exports: {},
          parents,children,
          hot: window.hotCreateModule(moduleId)
     };
     moreModules[moduleId].call(module.exports, module, module.exports,__webpack_require__);
     module.l = true;
    parents.forEach(parent=>{
      let parentModule = __webpack_require__.c[parent];
      parentModule.hot&&parentModule.hot._acceptedDependencies[moduleId]&&parentModule.hot._acceptedDependencies[moduleId]();
    });
    hotCurrentHash = currentHash;
  }
}
socket.on('connect',onConnected);
```

6.3 HotModuleReplacement.runtime.js

webpack\lib\HotModuleReplacement.runtime.js

```
function hotCreateModule(moduleId) {
        var hot = {
             _acceptedDependencies: {},
             accept: function(dep, callback) {
               for (var i = 0; i < dep.length; i++){
                 hot._acceptedDependencies[dep[i]] = callback;
               }
             }
       }
       return hot;
   }
```