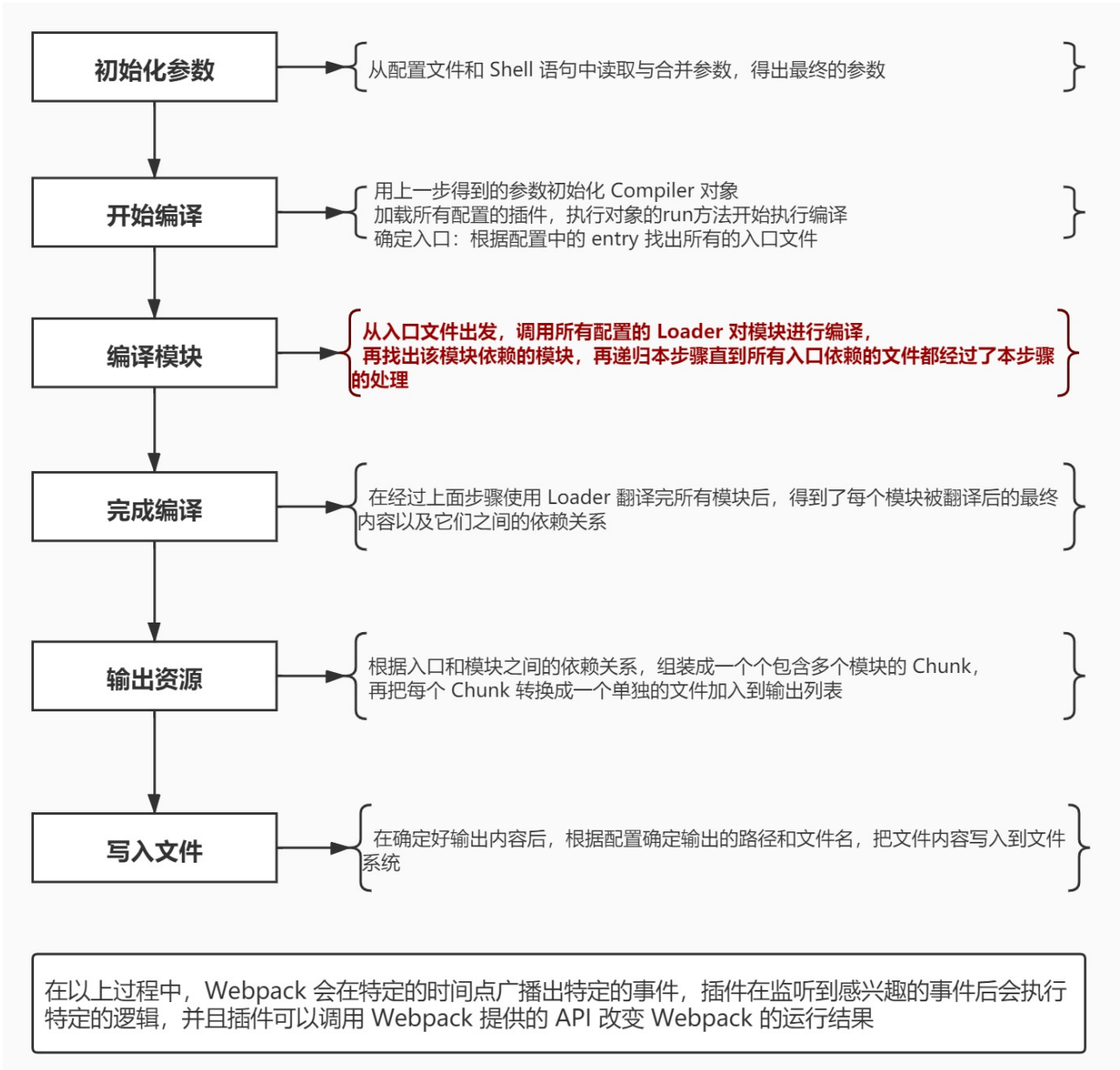
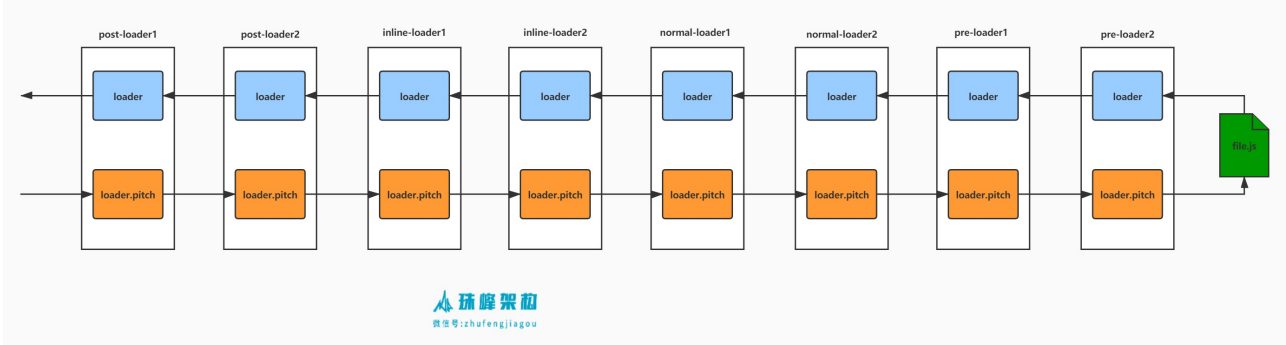
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=170 sentences=502, words=2551

1. inline loader、pre loader、 post loader和normal loader执行的先后顺序是什么？#

1.1 loader 运行的总体流程





1.2.loader-runner

1.2.1 loader 类型

- loader 的叠加顺序 (https://github.com/webpack/webpack/blob/v4.39.3/lib/NormalModuleFactory.js#L159-L339) = post(后置)+inline(内联)+normal(正常)+pre(前置)

1.2.2 特殊配置

- loaders/#configuration (https://webpack.js.org/concepts/loaders/#configuration)

符号 变量 含义 -!

noPreAutoLoaders 不要前置和普通 loader Prefixing with -! will disable all configured preLoaders and loaders but not postLoaders !

noAutoLoaders 不要普通 loader Prefixing with ! will disable all configured normal loaders ! !

noPrePostAutoLoaders 不要前后置和普通 loader,只要内联 loader Prefixing with !! will disable all configured loaders (preLoaders, loaders, postLoaders)

1.2.3 查找并执行

```
let path = require("path");
let nodeModules = path.resolve(__dirname, "node_modules");
let request = "-!inline-loader1!inline-loader2!./index.js";

let inlineLoaders = request
  .replace(/^-?!+/, "")
  .replace(/!!+/g, "!")
  .split("!");
let resource = inlineLoaders.pop();
let resolveLoader = (loader) => path.resolve(nodeModules, loader);

inlineLoaders = inlineLoaders.map(resolveLoader);
let rules = [
  {
    enforce: "pre",
    test: /\.css?$/,
    use: ["pre-loader1", "pre-loader2"],
  },
  {
    test: /\.css?$/,
    use: ["normal-loader1", "normal-loader2"],
  },
  {
    enforce: "post",
    test: /\.css?$/,
    use: ["post-loader1", "post-loader2"],
  },
];
let preLoaders = [];
let postLoaders = [];
let normalLoaders = [];
for (let i = 0; i < rules.length; i++) {
  let rule = rules[i];
  if (rule.test.test(resource)) {
    if (rule.enforce == "pre") {
      preLoaders.push(...rule.use);
    } else if (rule.enforce == "post") {
      postLoaders.push(...rule.use);
    } else {
      normalLoaders.push(...rule.use);
    }
  }
}
preLoaders = preLoaders.map(resolveLoader);
postLoaders = postLoaders.map(resolveLoader);
normalLoaders = normalLoaders.map(resolveLoader);

let loaders = [];

if (request.startsWith("!!")) {
  loaders = inlineLoaders;

} else if (request.startsWith("-!")) {
  loaders = [...postLoaders, ...inlineLoaders];

} else if (request.startsWith("!")) {
  loaders = [...postLoaders, ...inlineLoaders, ...preLoaders];
} else {
  loaders = [...postLoaders, ...inlineLoaders, ...normalLoaders, ...preLoaders];
}
console.log(loaders);
```

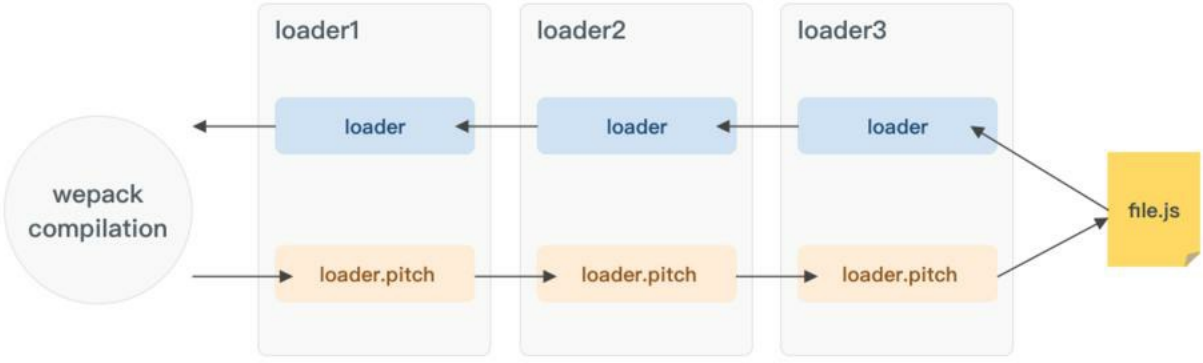
```
C:\aprepare\zhufengwebpackinterview\12.loader>node loader-runner.js
[
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\post-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\post-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\inline-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\inline-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\normal-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\normal-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\pre-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\pre-loader2'
]
pre2
pre1
normal2
normal1
inline2
inline1
post2
post1
null
{
  result: [
    "console.log('index.js');//pre2//pre1//normal2//normal1//inline2//inline1//post2//post1"
  ],
  resourceBuffer: <Buffer 63 6f 6e 73 6f 6c 65 2e 6c 6f 67 28 27 69 6e 64 65 78 2e 6a 73 27 29 3b>,
  cacheable: true,
  fileDependencies: [ 'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\src\\index.js' ],
  contextDependencies: []
}
```

1.2.4 pitch

- 比如 a!b!c!module, 正常调用顺序应该是 c、b、a，但是真正调用顺序是 a(pitch)、b(pitch)、c(pitch)、c、b、a,如果其中任何一个 pitching loader 返回了值就相当于在它以及它右边的 loader 已经执行完毕
- 比如如果 b 返回了字符串"result b", 接下来只有 a 会被系统执行，且 a 的 loader 收到的参数是 result b
- loader 根据返回值可以分为两种，一种是返回 js 代码（一个 module 的代码，含有类似 module.export 语句）的 loader，还有不能作为最左边 loader 的其他 loader
- 有时候我们想把两个第一种 loader chain 起来，比如 style-loader!css-loader! 问题是 css-loader 的返回值是一串 js 代码，如果按正常方式写 style-loader 的参数就是一串代码字符串
- 为了解决这种问题，我们需要在 style-loader 里执行 require(css-loader!resources)

pitch 与 loader 本身方法的执行顺序图

```
|- a-loader `pitch`
  |- b-loader `pitch`
    |- c-loader `pitch`
      |- requested module is picked up as a dependency
    |- c-loader normal execution
  |- b-loader normal execution
|- a-loader normal execution
```



1.2.1 loaders\loader1.js

loaders\loader1.js

```
function loader(source) {
  console.log("loader1", this.data);
  return source + "//loader1";
}
loader.pitch = function (remainingRequest, previousRequest, data) {
  data.name = "pitch1";
  console.log("pitch1");
};
module.exports = loader;
```

1.2.2 loaders\loader2.js

loaders\loader2.js

```
function loader(source) {
  console.log("loader2");
  return source + "//loader2";
}
loader.pitch = function (remainingRequest, previousRequest, data) {
  console.log("remainingRequest=", remainingRequest);
  console.log("previousRequest=", previousRequest);
  console.log("pitch2");

};
module.exports = loader;
```

1.2.3 loaders\loader3.js

loaders\loader3.js

```
function loader(source) {
  console.log("loader3");
  return source + "//loader3";
}
loader.pitch = function () {
  console.log("pitch3");
};
module.exports = loader;
```

1.3.4 webpack.config.js

```
module.exports = {
   resolveLoader: {
    alias: {
      'a-loader': path.resolve(__dirname, 'loaders/a.js')
    },
     modules: [path.resolve(__dirname, 'node_modules'),path.resolve(__dirname, 'loader')]
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        use: ['loader1', 'loader2', 'loader3']
      }
    ]
  }
}
```

```
C:\aprepare\zhufengwebpackinterview\12.loader>node loader-runner.js
[
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\post-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\post-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\inline-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\inline-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\normal-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\normal-loader2',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\pre-loader1',
  'C:\\aprepare\\zhufengwebpackinterview\\12.loader\\loaders\\pre-loader2'
]
normal1
inline2
inline1
post2
post1
null
{
  result: [ 'normal-loader2-pitch//normal1//inline2//inline1//post2//post1' ],
  resourceBuffer: null,
  cacheable: true,
  fileDependencies: [],
  contextDependencies: []
}
```

2. 是否写过Loader?描述一下编写loader的思路？

- babel-loader (https://github.com/babel/babel-loader/blob/master/src/index.js)
- @babel/core (https://babeljs.io/docs/en/next/babel-core.html)
- babel-plugin-transform-react-jsx (https://babeljs.io/docs/en/babel-plugin-transform-react-jsx/)
- previousRequest 前面的loader
- remainingRequest 后面的loader+资源路径
- data: 和普通的loader函数的第三个参数一样,而且loader执行的全程用的是同一个对象

属性 值 this.request /loaders/babel-loader.js!/src/index.js this.userRequest /src/index.js this.rawRequest ./src/index.js this.resourcePath /src/index.js

```
$ cnpm i @babel/preset-env @babel/core -D
```

```
const babel = require("@babel/core");
function loader(source, inputSourceMap,data) {

  const options = {
    presets: ["@babel/preset-env"],
    inputSourceMap: inputSourceMap,
    sourceMaps: true,
    filename: this.request.split("!")[1].split("/").pop(),
  };

  let { code, map, ast } = babel.transform(source, options);
  return this.callback(null, code, map, ast);
}
module.exports = loader;
```

```
resolveLoader: {
    alias: {
      "babel-loader": resolve('./build/babel-loader.js')
    },
    modules: [path.resolve('./loaders'), 'node_modules']
},
{
    test: /\.js$/,
    use:['babel-loader']
}
```

3. 是否写过Plugin?描述一下编写plugin的思路？#

3.1. plugin

插件向第三方开发者提供了 webpack 引擎中完整的能力。使用阶段式的构建回调，开发者可以引入它们自己的行为到 webpack 构建流程中。创建插件比创建 loader 更加高级，因为你将需要理解一些 webpack 底层的内部特性来做相应的钩子

3.1.1 为什么需要一个插件

- webpack 基础配置无法满足需求
- 插件几乎能够任意更改 webpack 编译结果
- webpack 内部也是通过大量内部插件实现的

3.1.2 可以加载插件的常用对象

对象 钩子
Compiler (https://github.com/webpack/webpack/blob/v4.39.3/lib/Compiler.js)

run,compile,compilation,make,emit,done
Compilation (https://github.com/webpack/webpack/blob/v4.39.3/lib/Compilation.js)

buildModule,normalModuleLoader,succeedModule,finishModules,seal,optimize,after-seal
Module Factory (https://github.com/webpack/webpack/blob/master/lib/ModuleFactory.js)

beforeResolver,afterResolver,module,parser Module
Parser (https://github.com/webpack/webpack/blob/master/lib/Parser.js)

] program,statement,call,expression
Template (https://github.com/webpack/webpack/blob/master/lib/Template.js)

hash,bootstrap,localVars,render

3.2. 创建插件

webpack 插件由以下组成：

- 一个 JavaScript 命名函数。
- 在插件函数的 prototype 上定义一个 apply 方法。
- 指定一个绑定到 webpack 自身的事件钩子。
- 处理 webpack 内部实例的特定数据。
- 功能完成后调用 webpack 提供的回调。

3.3. Compiler 和 Compilation

在插件开发中最重要的两个资源就是 `compiler`和 `compilation`对象。理解它们的角色是扩展 webpack 引擎重要的第一步。

- compiler 对象代表了完整的 webpack 环境配置。这个对象在启动 webpack 时被一次性建立，并配置好所有可操作的设置，包括 options，loader 和 plugin。当在 webpack 环境中应用一个插件时，插件将收到此 compiler 对象的引用。可以使用它来访问 webpack 的主环境。
- compilation 对象代表了一次资源版本构建。当运行 webpack 开发环境中间件时，每当检测到一个文件变化，就会创建一个新的 compilation，从而生成一组新的编译资源。一个 compilation 对象表现了当前的模块资源、编译生成资源、变化的文件、以及被跟踪依赖的状态信息。compilation 对象也提供了很多关键时机的回调，以供插件做自定义处理时选择使用。

3.4. 基本插件架构

- 插件是由「具有 apply 方法的 prototype 对象」所实例化出来的
- 这个 apply 方法在安装插件时，会被 webpack compiler 调用一次
- apply 方法可以接收一个 webpack compiler 对象的引用，从而可以在回调函数中访问到 compiler 对象

3.4.1 使用插件代码

- [使用插件] https://github.com/webpack/webpack/blob/master/lib/webpack.js#L60-L69 (https://github.com/webpack/webpack/blob/master/lib/webpack.js#L60-L69))

```
if (options.plugins && Array.isArray(options.plugins)) {
  for (const plugin of options.plugins) {
    plugin.apply(compiler);
  }
}
```

3.4.2 Compiler 插件

- done: new AsyncSeriesHook(["stats"]) (https://github.com/webpack/webpack/blob/master/lib/Compiler.js#L105)

3.4.2.1 同步

```
class DonePlugin {
  constructor(options) {
    this.options = options;
  }
  apply(compiler) {
    compiler.hooks.done.tap("DonePlugin", (stats) => {
      console.log("Hello ", this.options.name);
    });
  }
}
module.exports = DonePlugin;
```

3.4.2.2 异步

```
class DonePlugin {
  constructor(options) {
    this.options = options;
  }
  apply(compiler) {
    compiler.hooks.done.tapAsync("DonePlugin", (stats, callback) => {
      console.log("Hello ", this.options.name);
      callback();
    });
  }
}
module.exports = DonePlugin;
```

3.4.3 使用插件

- 要安装这个插件，只需要在你的 webpack 配置的 plugin 数组中添加一个实例

```
const DonePlugin = require("./plugins/DonePlugin");
module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.resolve("build"),
    filename: "bundle.js",
  },
  plugins: [new DonePlugin({ name: "zhufeng" })],
};
```

4. webpack打包的原理是什么?聊一聊babel和抽象语法树吧

- astexplorer (https://astexplorer.net/)

4.1 index.js

```
let title = require('./title.js');
console.log(title);
```

4.2 title.js

```
module.exports = 'title';
```

4.3 packer.js

```
const fs = require("fs");
const path = require("path");
const types = require("babel-types");
const parser = require("@babel/parser");
const traverse = require("@babel/traverse").default;
const generate = require("@babel/generator").default;
const baseDir = process.cwd().replace(/\\/g,path.posix.sep);
const entry = path.posix.join(baseDir, "src/index.js");
let modules = [];
function buildModule(absolutePath){
  const body = fs.readFileSync(absolutePath, "utf-8");
  const ast = parser.parse(body, {
    sourceType: "module",
  });
  const moduleId = "./" + path.posix.relative(baseDir, absolutePath);
  const module = { id: moduleId };
  const deps = [];
  traverse(ast, {
    CallExpression({ node }) {
      if (node.callee.name === 'require') {
        node.callee.name = "__webpack_require__";
        let moduleName = node.arguments[0].value;
        const dirname = path.posix.dirname(absolutePath);
        const depPath = path.posix.join(dirname, moduleName);
        const depModuleId = "./" + path.posix.relative(baseDir, depPath);
        node.callee.name = "__webpack_require__";
        node.arguments = [types.stringLiteral(depModuleId)];
        deps.push(buildModule(depPath));
      }
    }
  });
  let { code } = generate(ast);
  module._source = code;
  module.deps = deps;
  modules.push(module);
  return module;
}
let entryModule = buildModule(entry);
let content = `
(function (modules) {
    function __webpack_require__(moduleId) {
        var module = {
            i: moduleId,
            exports: {}
        };
        modules[moduleId].call(
            module.exports,
            module,
            module.exports,
            __webpack_require__
        );
        return module.exports;
    }

    return __webpack_require__("${entryModule.id}");
})(
    {
      ${modules
        .map(
          (module) =>
            `"${module.id}": function (module, exports,__webpack_require__) {${module._source}}`
        )
        .join(",")}
    }
);
`;

fs.writeFileSync("./dist/bundle.js", content);
```

5. tree-shaking了解过么?它的实现原理说一下

```
var babel = require("@babel/core");
let { transform } = require("@babel/core");
```

5.1 实现按需加载

- lodashjs (https://www.lodashjs.com/docs/4.17.5.html#concat)
- babel-core (https://babeljs.io/docs/en/babel-core)
- babel-plugin-import (https://www.npmjs.com/package/babel-plugin-import)

```
import { flatten, concat } from "lodash";
```

```
- ImportDeclaration {
  - specifiers: [
    - ImportSpecifier {
      - imported: Identifier {
          name: "flatten"
        }
      - local: Identifier {
          name: "flatten"
        }
      }
    - ImportSpecifier {
      - imported: Identifier {
          name: "concat"
        }
      - local: Identifier = $node {
          name: "concat"
        }
      }
    ]
    importKind: "value"
  - source: StringLiteral {
    - extra: {
        rawValue: "lodash"
        raw: "\"lodash\""
      }
      value: "lodash"
    }
  }
```

转换为

```
import flatten from "lodash/flatten";
import concat from "lodash/flatten";
```

```
ImportDeclaration {
  - specifiers: [
    - ImportDefaultSpecifier {
      - local: Identifier {
          name: "flatten"
        }
      }
    ]
    importKind: "value"
  - source: StringLiteral {
    - extra: {
        rawValue: "lodash/flatten"
        raw: "\"lodash/flatten\""
      }
      value: "lodash/flatten"
    }
}
```

5.2 webpack 配置 #

```
cnpm i webpack webpack-cli -D
```

```
const path = require("path");
module.exports = {
  mode: "development",
  entry: "./src/index.js",
  output: {
    path: path.resolve("dist"),
    filename: "bundle.js",
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        use: {
          loader: "babel-loader",
          options: {
            plugins: [["import", { library: "lodash" }]],
          },
        },
      },
    ],
  },
};
```

> 编译顺序为首先 plugins从左往右,然后 presets从右往左

5.3 babel 插件

- babel-plugin-import.js放置在 node_modules 目录下

```
let babel = require("@babel/core");
let types = require("babel-types");
const visitor = {
  ImportDeclaration: {
    enter(path, state = { opts }) {
      const specifiers = path.node.specifiers;
      const source = path.node.source;
      if (
        state.opts.library == source.value &&
        !types.isImportDefaultSpecifier(specifiers[0])
      ) {
        const declarations = specifiers.map((specifier, index) => {
          return types.ImportDeclaration(
            [types.importDefaultSpecifier(specifier.local)],
            types.stringLiteral(`${source.value}/${specifier.local.name}`)
          );
        });
        path.replaceWithMultiple(declarations);
      }
    },
  },
};
module.exports = function (babel) {
  return {
    visitor,
  };
};
```

6. webpack的热更新是如何做到的?说明其原理?

6.1. 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

6.2. 搭建HMR项目

6.2.1 安装依赖的模块

```
cnpm i webpack webpack-cli webpack-dev-server mime html-webpack-plugin express socket.io events -S
```

6.2.2 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"
  }
}
```

6.2.3 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'
        })
    ]
}
```

6.2.4 src\index.js

src\index.js

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

6.2.5 src\title.js

src\title.js

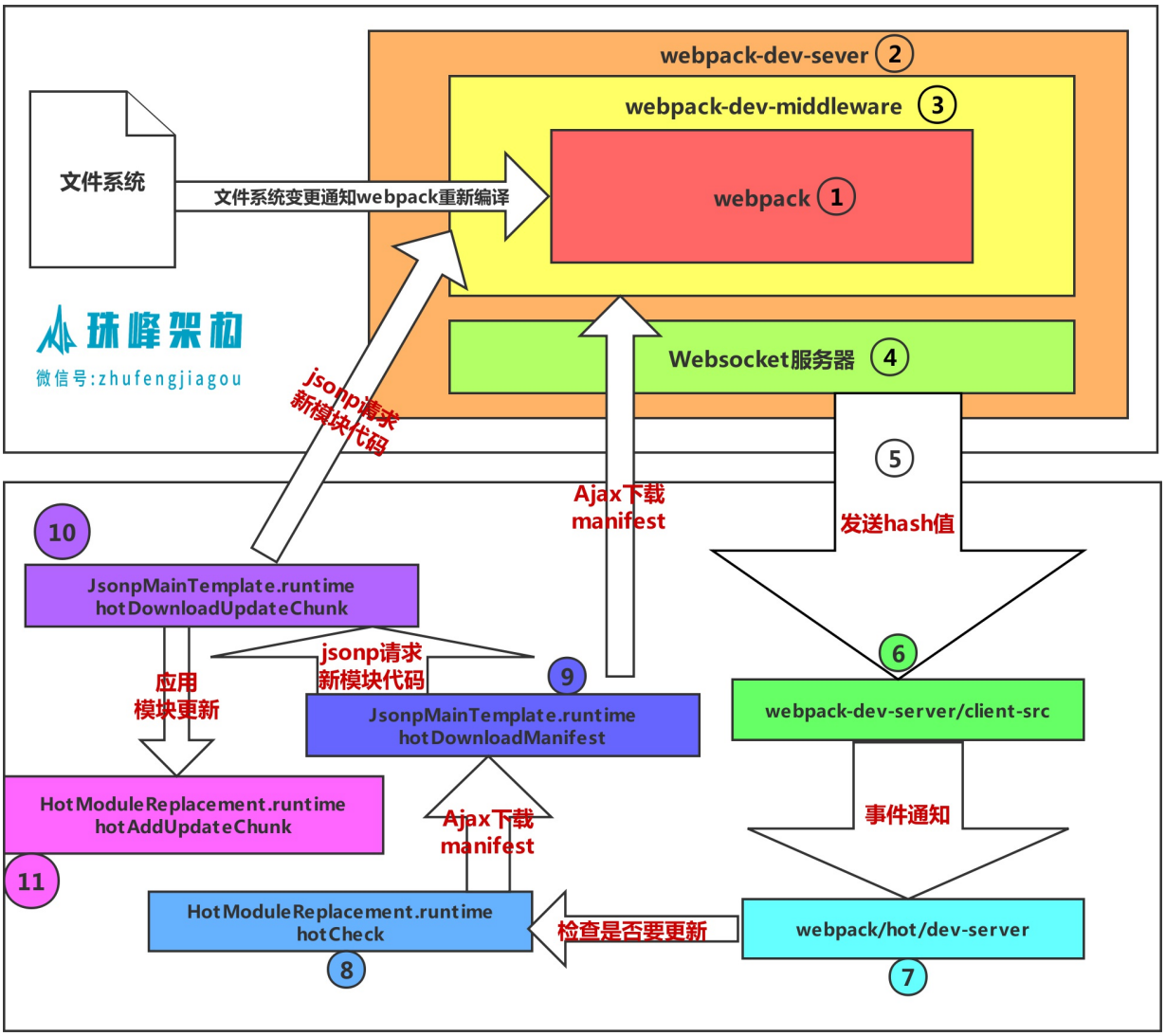
```
export default 'hello';
```

6.2.6 src\index.html

src\index.html

```
    webpack热更新
```

6.3.流程图



6.4.实现

6.4.1 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: {
    hot: true,
    contentBase: path.join(__dirname, "dist"),
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
      filename: "index.html",
    }),
    new webpack.HotModuleReplacementPlugin()
  ]
};
```

6.4.2 index.js

src\index.js

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

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

6.4.3 src\title.js

src\title.js

```
module.exports = 'title7';
```

6.4.4 src\index.html

src\index.html

```
  webpack热更新
```

6.4.5 webpack-dev-server.js

webpack-dev-server.js

```
const path = require("path");
const fs = require("fs");
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) {

    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({}, (err) => {
      console.log("编译成功");
    });

    const webpackDevMiddleware = (req, res, next) => {
      if (req.url === "/favicon.ico") {
        return res.sendStatus(404);
      }
      let filename = path.join(config.output.path, req.url.slice(1));
      try {
        let stats = fs.statSync(filename);
        if (stats.isFile()) {
          let content = fs.readFileSync(filename);
          res.header("Content-Type", mime.getType(filename));
          res.send(content);
        } else {
          next();
        }
      } catch (error) {
          return res.sendStatus(404);
      }
    };
    app.use(webpackDevMiddleware);
    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(8080);
```

6.4.6 webpackHotDevClient.js

webpackHotDevClient.js

```
let socket = io("/");
let currentHash;
let hotCurrentHash;
const onConnected = () => {
  console.log("客户端已经连接");

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

  socket.on("ok", () => {
    hotCheck();
  });
  socket.on("disconnect", () => {
     hotCurrentHash = currentHash = null;
  });
};

function hotCheck() {
  if (!hotCurrentHash || hotCurrentHash === currentHash) {
    return (hotCurrentHash = currentHash);
  }

  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() {
  var url = "/" + hotCurrentHash + ".hot-update.json";
  return fetch(url).then(res => res.json()).catch(error=>{console.log(error);});
}

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,
      exports: {},
      parents,
      children,
      hot: window.hotCreateModule(),
    });
    moreModules[moduleId].call(
      module.exports,
      module,
      module.exports,
      __webpack_require__
    );
    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);
window.hotCreateModule = () => {
  var hot = {
    _acceptedDependencies: {},
    _acceptedDependencies: function (dep, callback) {
      for (var i = 0; i < dep.length; i++) {
        hot._acceptedDependencies[dep[i]] = callback;
      }
    },
  };
  return hot;
}
```

7.从零实现Webpack5模块联邦原理并实现微前端

- dll、external、npm包、umd、qiankun微前端等代码共享方案缺点分析
- webpack5中最激动人心的新特性ModuleFederation实战和原理
- webpack5的ModuleFederation微前端实战
- 从零实现webpack5 ModuleFederation