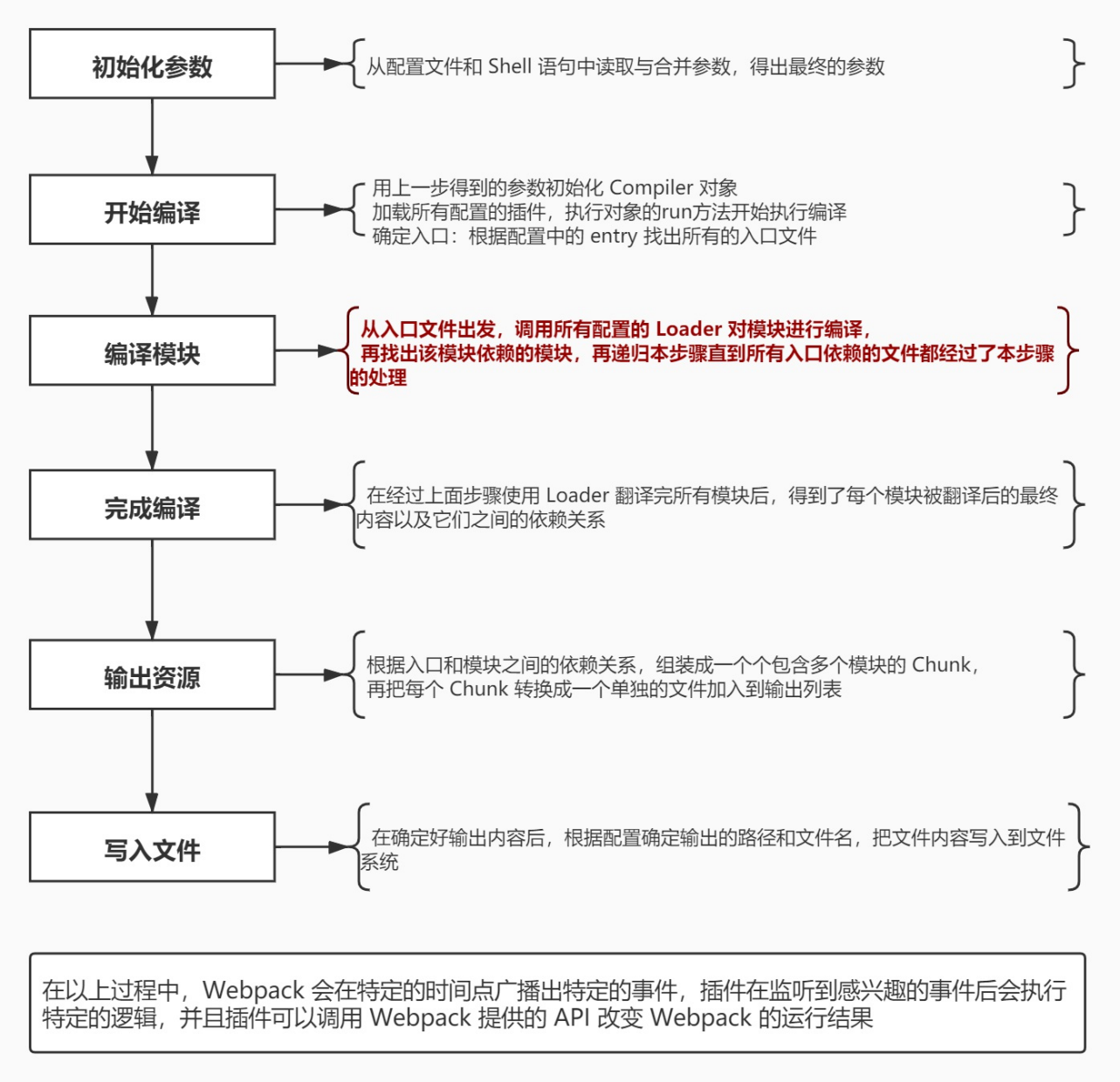
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=115 sentences=460, words=2358

1.loader

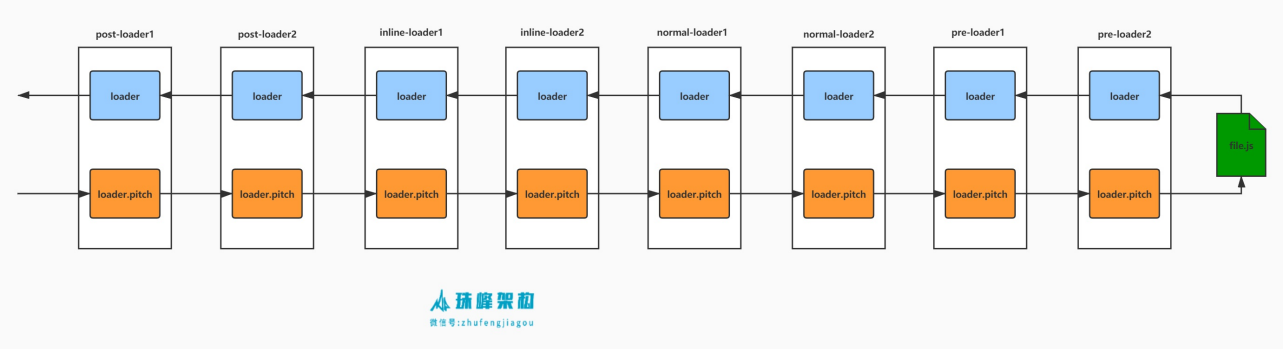
- 所谓 loader 只是一个导出为函数的 JavaScript 模块。它接收上一个 loader 产生的结果或者资源文件(resource file)作为入参。也可以用多个 loader 函数组成 loader chain
- compiler 需要得到最后一个 loader 产生的处理结果。这个处理结果应该是 String 或者 Buffer（被转换为一个 string）

1.1 loader 运行的总体流程



1.2 loader-runner

- loader-runner (https://github.com/webpack/loader-runner#readme)是一个执行 loader 链条的的模块



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 执行流程

1.2.2.1 runner.js

```
const { runLoaders } = require("loader-runner");
const path = require("path");
const fs = require("fs");
const entryFile = path.resolve(__dirname, "src/index.js");

let request = `inline-loader1!inline-loader2!${entryFile}`;
let rules = [
  {
    test: /\.js$/,
    use: ["normal-loader1", "normal-loader2"],
  },
  {
    test: /\.js$/,
    enforce: "post",
    use: ["post-loader1", "post-loader2"],
  },
  {
    test: /\.js$/,
    enforce: "pre",
    use: ["pre-loader1", "pre-loader2"],
  },
];
let parts = request.replace(/^-?!+/, "").split("!");
let resource = parts.pop();
let inlineLoaders = parts;
let preLoaders = [],
  postLoaders = [],
  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);
    }
  }
}
let loaders = [
  ...postLoaders,
  ...inlineLoaders,
  ...normalLoaders,
  ...preLoaders,
];
let resolveLoader = (loader) =>
  path.resolve(__dirname, "loaders-chain", loader);

loaders = loaders.map(resolveLoader);
runLoaders(
  {
    resource,
    loaders,
    context: { name: "zhufeng", age: 100 },
    readResource: fs.readFile.bind(this),
  },
  (err, result) => {
    console.log(err);
    console.log(result);
    console.log(
      result.resourceBuffer ? result.resourceBuffer.toString("utf8") : null
    );
  }
);
```

1.2.2.2 pre-loader1.js

loaders\pre-loader1.js

```
function loader(source) {
  console.log("pre1");
  return source + "//pre1";
}
module.exports = loader;
```

1.2.2.3 pre-loader2.js

loaders\pre-loader2.js

```
function loader(source) {
  console.log("pre2");
  return source + "//pre2";
}
module.exports = loader;
```

1.2.2.4 normal-loader1.js

loaders\normal-loader1.js

```
function loader(source) {
  console.log("normal1");
  return source + "//normal1";
}
loader.pitch = function () {
  return "normal1pitch";
};
module.exports = loader;
```

1.2.2.5 normal-loader2.js

loaders\normal-loader2.js

```
function loader(source) {
  console.log("normal2");
  return source + "//normal2";
}

module.exports = loader;
```

1.2.2.6 inline-loader1.js

loaders\inline-loader1.js

```
function loader(source) {
  console.log("inline1");
  return source + "//inline1";
}

module.exports = loader;
```

1.2.2.7 inline-loader2.js

loaders\inline-loader2.js

```
function loader(source) {
  console.log("inline2");
  return source + "//inline2";
}
module.exports = loader;
```

1.2.2.8 post-loader1.js

loaders\post-loader1.js

```
function loader(source) {
  console.log("post1");
  return source + "//post1";
}
module.exports = loader;
```

1.2.2.9 post-loader2.js

loaders\post-loader2.js

```
function loader(source) {
  console.log("post2");
  return source + "//post2";
}
module.exports = loader;
```

```
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: []
}
```

1.3 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)

```
const { runLoaders } = require("./loader-runner");
const path = require("path");
const fs = require("fs");//webpack-dev-server启开发服务器的时候 memory-fs
const entryFile = path.resolve(__dirname, "src/index.js");
//如何配置行内
let request = `inline-loader1!inline-loader2!${entryFile}`;
let rules = [
  {
    test: /\.js$/,
    use: ["normal-loader1", "normal-loader2"],
  },
  {
    test: /\.js$/,
    enforce: "post",
    use: ["post-loader1", "post-loader2"],
  },
  {
    test: /\.js$/,
    enforce: "pre",
    use: ["pre-loader1", "pre-loader2"],
  },
];
let parts = request.replace(/^-?!+/,'').split('!');
let resource = parts.pop();//弹出最后一个元素 entryFile=src/index.js
let inlineLoaders = parts;//[inline-loader1,inline-loader2]
let preLoaders = [],postLoaders=[],normalLoaders=[];
for(let i=0;i+let loaders = [];
+if(request.startsWith('!!')){
+    loaders = [...inlineLoaders];
+    //noPreAutoLoaders
+}else if(request.startsWith('-!')){
+    loaders = [...postLoaders,...inlineLoaders];
+}else if(request.startsWith('!')){
+    //noAutoLoaders
+    loaders = [...postLoaders,...inlineLoaders,...preLoaders];
+}else{
+    loaders = [...postLoaders,...inlineLoaders,...normalLoaders,...preLoaders];
+}
let resolveLoader = loader=>path.resolve(__dirname,'loaders-chain',loader)
//把loader数组从名称变成绝对路径
loaders= loaders.map(resolveLoader);
runLoaders({
    resource,//你要加载的资源
    loaders,
    context:{name:'zhufeng',age:100},//保存一些状态和值
    readResource:fs.readFile.bind(this)
},(err,result)=>{
    console.log(err);//运行错误
    console.log(result);//运行的结果
    console.log(result.resourceBuffer?result.resourceBuffer.toString('utf8'):null);//读到的原始的文件
});
```

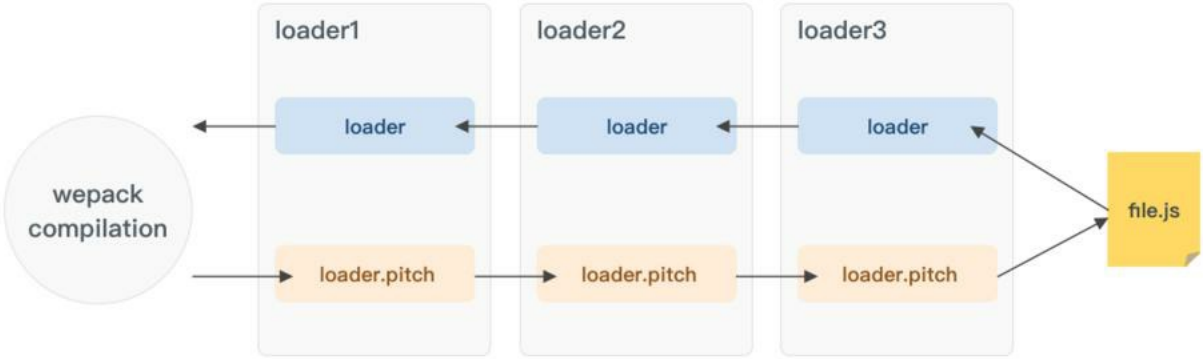
1.4 pitch

- 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
```



2.babel-loader

- 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
- currentRequest 自己和后面的 loader+资源路径
- 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
- 注意 sourceMaps最后有个 s

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

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

```
const core = require("@babel/core");
const path = require("path");
function loader(source) {
  let filename = this.resourcePath.split(path.sep).pop();
  let options = this.getOptions();
  let loaderOptions = {
    ...options,
    sourceMaps: true,
    filename,
  };

  let { code, map, ast } = core.transformSync(source, loaderOptions);

  this.callback(null, code, map, ast);
}
module.exports = loader;
```

webpack.config.js

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");
module.exports = {
  mode: "development",
  devtool: "source-map",
  entry: "./src/index.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "[name].js",
  },
  devServer: {
    hot: false,
  },
  resolveLoader: {
    alias: {
      "babel-loader": path.resolve(__dirname, "loader/babel-loader.js"),
    },
    modules: [path.resolve("./loader"), "node_modules"],
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
          options: {
            presets: ["@babel/preset-env"],
          },
        },
      },
      {
        test: /\.less$/,
        exclude: /node_modules/,
        use: ["style-loader", "less-loader"],
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./src/index.html",
    }),
  ],
};
```

3. style-loader

- css-loader (https://github.com/webpack-contrib/css-loader/blob/master/lib/loader.js) 的作用是处理 css 中的 @import 和 url 这样的外部资源
- style-loader (https://github.com/webpack-contrib/style-loader/blob/master/index.js) 的作用是把样式插入到 DOM 中，方法是在 head 中插入一个 style 标签，并把样式写入到这个标签的 innerHTML 里
- less-loader (https://github.com/webpack-contrib/less-loader) 把 less 编译成 css
- pitching-loader (https://webpack.js.org/api/loaders/#pitching-loader)
- loader-utils (https://github.com/webpack/loader-utils)
- !! (https://webpack.js.org/concepts/loaders/#configuration)

3.1 安装依赖

```
$ npm i less  -D
```

3.2 使用 less-loader

3.2.1 index.js

src\index.js

```
import "./index.less";
```

3.2.2 src\index.less

src\index.less

```
@color: red;
#root {
  color: @color;
}
```

3.2.3 src\index.html

src\index.html

```
<div id="root">rootdiv>
```

3.2.4 webpack.config.js

webpack.config.js

```
{
  test: /\.less$/,
  use: [
    'style-loader',
    'less-loader'
  ]
}
```

3.2.5 less-loader.js

```
let less = require("less");
function loader(source) {
  let callback = this.async();
  less.render(source, { filename: this.resource }, (err, output) => {
    callback(err, output.css);
  });
}
module.exports = loader;
```

3.2.6 style-loader

```
function loader(source) {
  let script = `
      let style = document.createElement("style");
      style.innerHTML = ${JSON.stringify(source)};
    document.head.appendChild(style);
    `;
  return script;
}
module.exports = loader;
```

3.3 两个左侧模块连用

3.3.1 less-loader.js

```
let less = require("less");
function loader(source) {
  let callback = this.async();
  less.render(source, { filename: this.resource }, (err, output) => {
    callback(err, `module.exports = ${JSON.stringify(output.css)}`);
  });
}
module.exports = loader;
```

3.3.2 style-loader.js

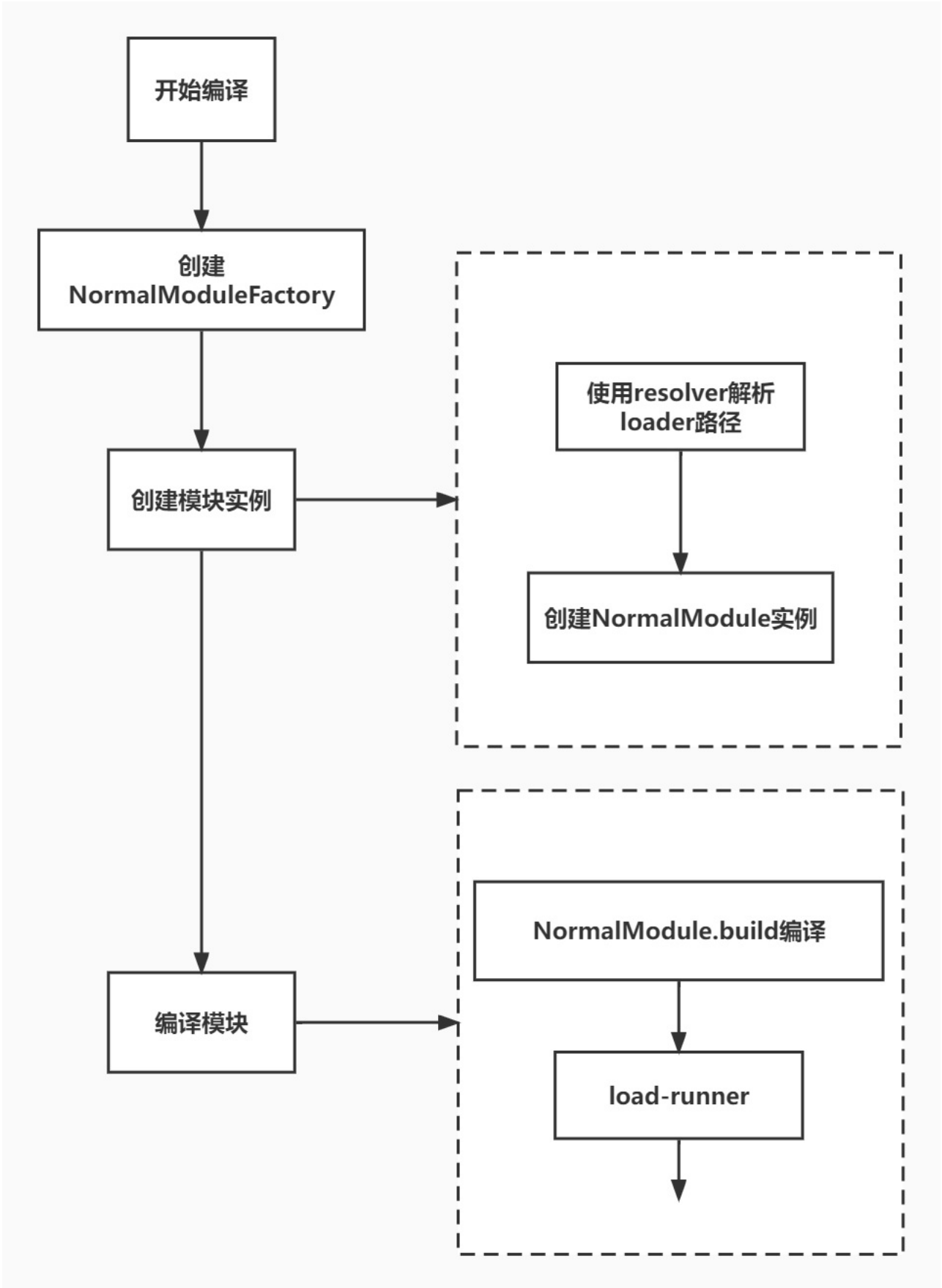
```
const path = require("path");
function loader() {}
loader.pitch = function (remainingRequest) {

  const request = "!!"+remainingRequest.split('!').map(requestPath=>{
    return this.utils.contextify(this.context,requestPath)
  }).join('!');
  let script = `
      let style = require("${request}");
      let style = document.createElement('style');
      style.innerHTML = require(${stringifyRequest(
        this,
        "!!" + remainingRequest
      )});
      document.head.appendChild(style);
    `;
  return script;
};
loader.pitch = function (remainingRequest) {

  let script = `
      let style = document.createElement('style');
      style.innerHTML = require(${stringifyRequest(
        this,
        "!!" + remainingRequest
      )});
      document.head.appendChild(style);
    `;
  console.log(script);
  return script;
};
function stringifyRequest(loaderContext, request) {
  const splitted = request.split("!");
  const { context } = loaderContext;
  return JSON.stringify(
      splitted
        .map((part) => {
          part = path.relative(context, part);
          if (part[0] !== ".") part = "./" + part;
          return part.replace(/\\/g, "/");
        })
        .join("!")
  );
}
function stringifyRequest(loaderContext, request) {
  let prefixRep = /^-?!+/;
  let prefixResult = request.match(prefixRep);
  let prefix = prefixResult ? prefixResult[0] : "";
  const splitted = request.replace(prefixRep, "").split("!");
  const { context } = loaderContext;
  return JSON.stringify(
    prefix +
      splitted
        .map((part) => {
          part = path.relative(context, part);
          if (part[0] !== ".") part = "./" + part;
          return part.replace(/\\/g, "/");
        })
        .join("!")
  );
}
module.exports = loader;
```

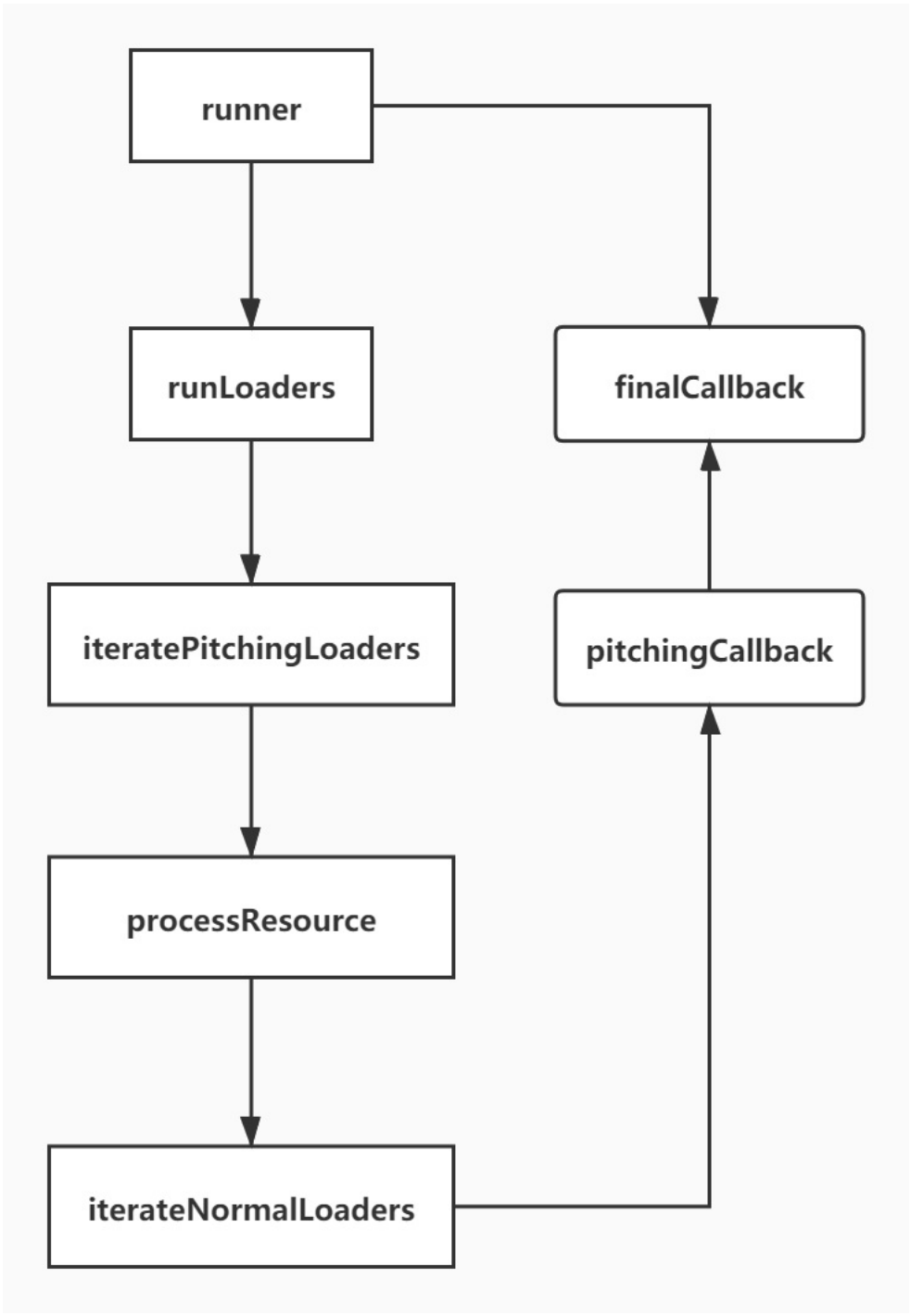
4.loader原理

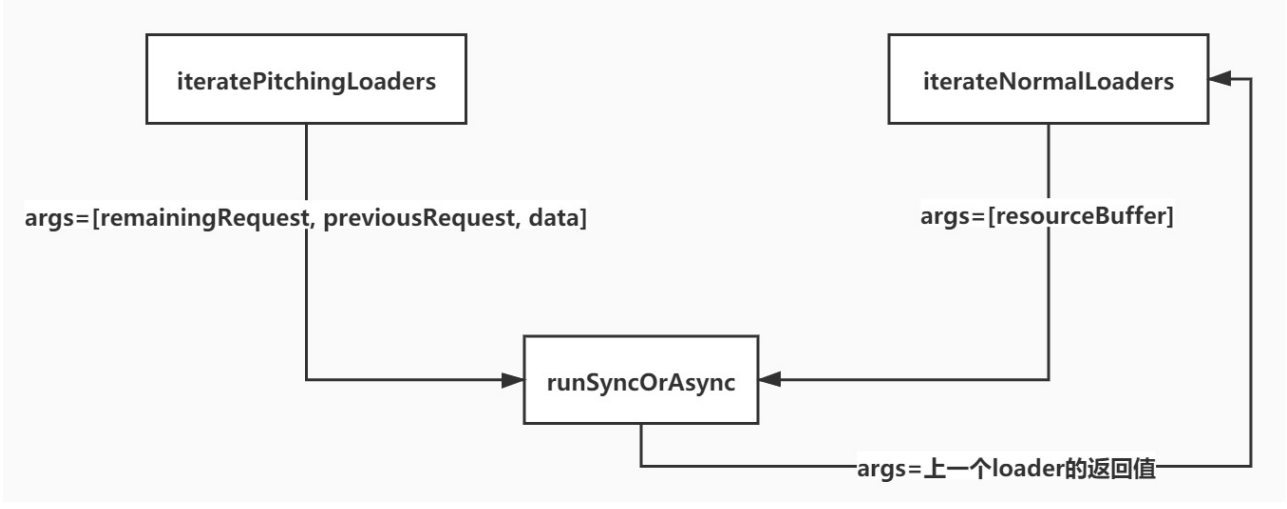
4.1 loader运行流程



4. loader-runner 实现 #

- LoaderRunner.js (https://github.com/webpack/loader-runner/blob/v2.4.0/lib/LoaderRunner.js)
- NormalModuleFactory.js (https://github.com/webpack/webpack/blob/v4.39.3/lib/NormalModuleFactory.js#L180)
- NormalModule.js (https://github.com/webpack/webpack/blob/v4.39.3/lib/NormalModule.js#L292)


runner
runLoaders
finalCallback
iteratePitchingLoaders
pitchingCallback
processResource
iterateNormalLoaders


iteratePitchingLoaders
iterateNormalLoaders
args=[remainingRequest, previousRequest, data]
args=[resourceBuffer]
runSyncOrAsync
args=上一个loader的返回值

```
let fs = require("fs");

function createLoaderObject(loader) {
  let normal = require(loader);
  let pitch = normal.pitch;
  let raw = normal.raw;
  return {
    path: loader,
    normal,
    pitch,
    raw,
    data: {},
    pitchExecuted: false,
    normalExecuted: false,
  };
}
function convertArgs(args, raw) {
  if (raw && !Buffer.isBuffer(args[0])) {
    args[0] = Buffer.from(args[0]);
  } else if (!raw && Buffer.isBuffer(args[0])) {
    args[0] = args[0].toString("utf8");
  }
}
function iterateNormalLoaders(
  processOptions,
  loaderContext,
  args,
  pitchingCallback
) {
  if (loaderContext.loaderIndex < 0) {
    return pitchingCallback(null, args);
  }
  let currentLoader = loaderContext.loaders[loaderContext.loaderIndex];
  if (currentLoader.normalExecuted) {
    loaderContext.loaderIndex--;
    return iterateNormalLoaders(
      processOptions,
      loaderContext,
      args,
      pitchingCallback
    );
  }
  let fn = currentLoader.normal;
  currentLoader.normalExecuted = true;
  convertArgs(args, currentLoader.raw);
  runSyncOrAsync(fn, loaderContext, args, (err, ...returnArgs) => {
    if (err) return pitchingCallback(err);
    return iterateNormalLoaders(
      processOptions,
      loaderContext,
      returnArgs,
      pitchingCallback
    );
  });
}
function processResource(processOptions, loaderContext, pitchingCallback) {
  processOptions.readResource(loaderContext.resource, (err, resourceBuffer) => {
    processOptions.resourceBuffer = resourceBuffer;
    loaderContext.loaderIndex--;
    iterateNormalLoaders(
      processOptions,
      loaderContext,
      [resourceBuffer],
      pitchingCallback
    );
  });
}
function iteratePitchingLoaders(
  processOptions,
  loaderContext,
  pitchingCallback
) {

  if (loaderContext.loaderIndex >= loaderContext.loaders.length) {
    return processResource(processOptions, loaderContext, pitchingCallback);
  }
  let currentLoader = loaderContext.loaders[loaderContext.loaderIndex];
  if (currentLoader.pitchExecuted) {
    loaderContext.loaderIndex++;
    return iteratePitchingLoaders(
      processOptions,
      loaderContext,
      pitchingCallback
    );
  }
  let fn = currentLoader.pitch;
  currentLoader.pitchExecuted = true;
  if (!fn) {
    return iteratePitchingLoaders(
      processOptions,
      loaderContext,
      pitchingCallback
    );
  }

  runSyncOrAsync(
    fn,
    loaderContext,
    [
      loaderContext.remainingRequest,
      loaderContext.previousRequest,
      loaderContext.data,
    ],
```

```
    (err, ...args) => {

      if (args.length > 0 && args.some((item) => item)) {
        loaderContext.loaderIndex--;
        iterateNormalLoaders(
          processOptions,
          loaderContext,
          args,
          pitchingCallback
        );
      } else {
        return iteratePitchingLoaders(
          processOptions,
          loaderContext,
          pitchingCallback
        );
      }
    }
  );
}
function runSyncOrAsync(fn, loaderContext, args, runCallback) {
  let isSync = true;
  loaderContext.callback = (...args) => {
    runCallback(null, ...args);
  };
  loaderContext.async = () => {
    isSync = false;
    return loaderContext.callback;
  };

  let result = fn.apply(loaderContext, args);
  if (isSync) {

    runCallback(null, result);
  }
}
function runLoaders(options, finalCallback) {
  let {
    resource,
    loaders = [],
    context = {},
    readResource = fs.readFile,
  } = options;
  let loaderObjects = loaders.map(createLoaderObject);
  let loaderContext = context;
  loaderContext.resource = resource;
  loaderContext.readResource = readResource;
  loaderContext.loaders = loaderObjects;
  loaderContext.loaderIndex = 0;
  loaderContext.callback = null;
  loaderContext.async = null;

  Object.defineProperty(loaderContext, "request", {
    get() {

      return loaderContext.loaders
        .map((loader) => loader.path)
        .concat(loaderContext.resource)
        .join("!");
    },
  });

  Object.defineProperty(loaderContext, "remainingRequest", {
    get() {

      return loaderContext.loaders
        .slice(loaderContext.loaderIndex + 1)
        .map((loader) => loader.path)
        .concat(loaderContext.resource)
        .join("!");
    },
  });

  Object.defineProperty(loaderContext, "currentRequest", {
    get() {

      return loaderContext.loaders
        .slice(loaderContext.loaderIndex)
        .map((loader) => loader.path)
        .concat(loaderContext.resource)
        .join("!");
    },
  });

  Object.defineProperty(loaderContext, "previousRequest", {
    get() {

      return loaderContext.loaders
        .slice(0, loaderContext.loaderIndex)
        .map((loader) => loader.path)
        .join("!");
    },
  });
  Object.defineProperty(loaderContext, "data", {
    get() {

      return loaderContext.loaders[loaderContext.loaderIndex].data;
    },
  });
  let processOptions = {
    resourceBuffer: null,
    readResource,
  };
  iteratePitchingLoaders(processOptions, loaderContext, (err, result) => {
```

```
    finalCallback(err, {
      result,
      resourceBuffer: processOptions.resourceBuffer,
    });
  });
}
exports.runLoaders = runLoaders;
```