打包分析和webpack源码

分析一：打包的时间分析

- 如果我们希望看到每一个loader、每一个Plugin消耗的打包时间，可以借助于一个插件：speed-measure-webpack-plugin
  - 注意：该插件在最新的webpack版本中存在一些兼容性的问题（和部分Plugin不兼容）
  - 截止2021-3-10日，但是目前该插件还在维护，所以可以等待后续是否更新；
  - 我这里暂时的做法是把不兼容的插件先删除掉，也就是不兼容的插件不显示它的打包时间就可以了；
- 第一步，安装speed-measure-webpack-plugin插件

```
npm install speed-measure-webpack-plugin -D
```

- 第二步，使用speed-measure-webpack-plugin插件
  - 创建插件导出的对象 SpeedMeasurePlugin；
  - 使用 smp.wrap 包裹我们导出的webpack配置；

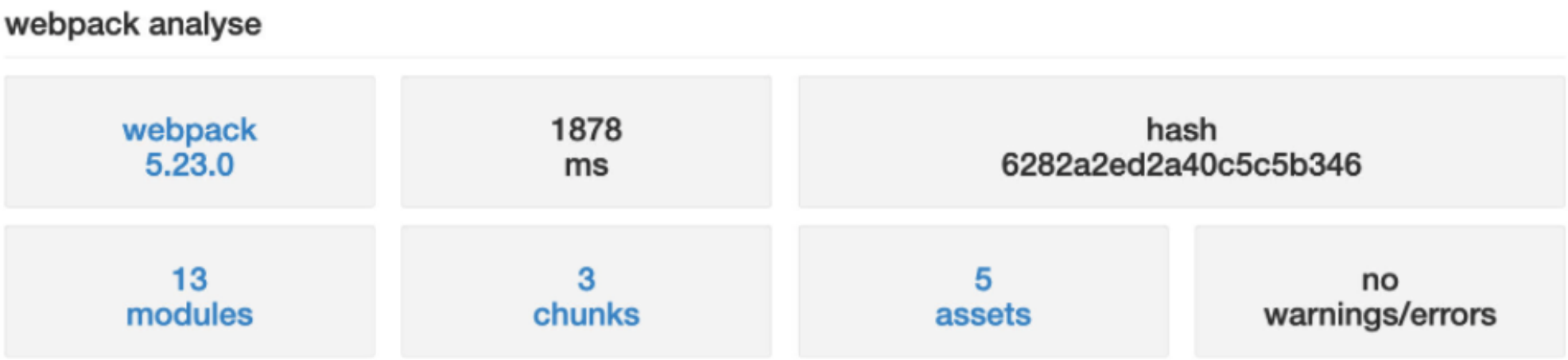
```
const SpeedMeasurePlugin = require("speed-measure-webpack-plugin");
const smp = new SpeedMeasurePlugin();

const webpackConfig = smp.wrap({
  plugins: [new MyPlugin(), new MyOtherPlugin()],
});
```

分析二：打包后文件分析

- 方案一：生成一个stats.json的文件

```
"buiebpack ld:stats": "w--config ./config/webpack.common.js --env production --profile --json=stats.json",
```

- 通过执行npm run build:status可以获取到一个stats.json的文件：
  - 这个文件我们自己分析不容易看到其中的信息；
  - 可以放到 http://webpack.github.com/analyse，进行分析



分析二：打包后文件分析

- 方案二：使用webpack-bundle-analyzer工具
  - 另一个非常直观查看包大小的工具是webpack-bundle-analyzer。
- 第一步，我们可以直接安装这个工具：

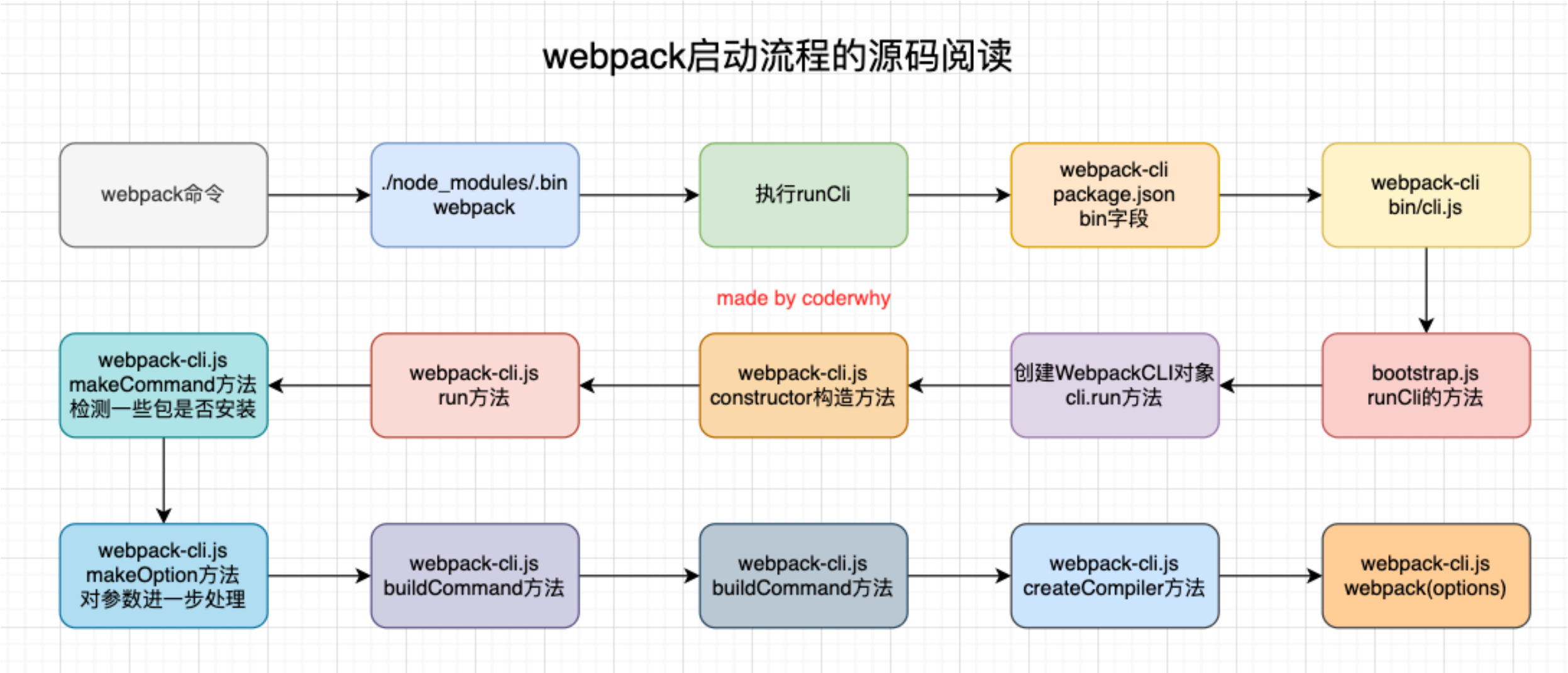
```
npm install webpack-bundle-analyzer -D
```

- 第二步，我们可以在webpack配置中使用该插件：

```
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;

module.exports = {
  plugins: [
    new BundleAnalyzerPlugin()
  ]
}
```

- 在打包webpack的时候，这个工具是帮助我们打开一个8888端口上的服务，我们可以直接的看到每个包的大小。
  - 比如有一个包时通过一个Vue组件打包的，但是非常的大，那么我们可以考虑是否可以拆分出多个组件，并且对其进行懒加载；
  - 比如一个图片或者字体文件特别大，是否可以对其进行压缩或者其他的优化处理；

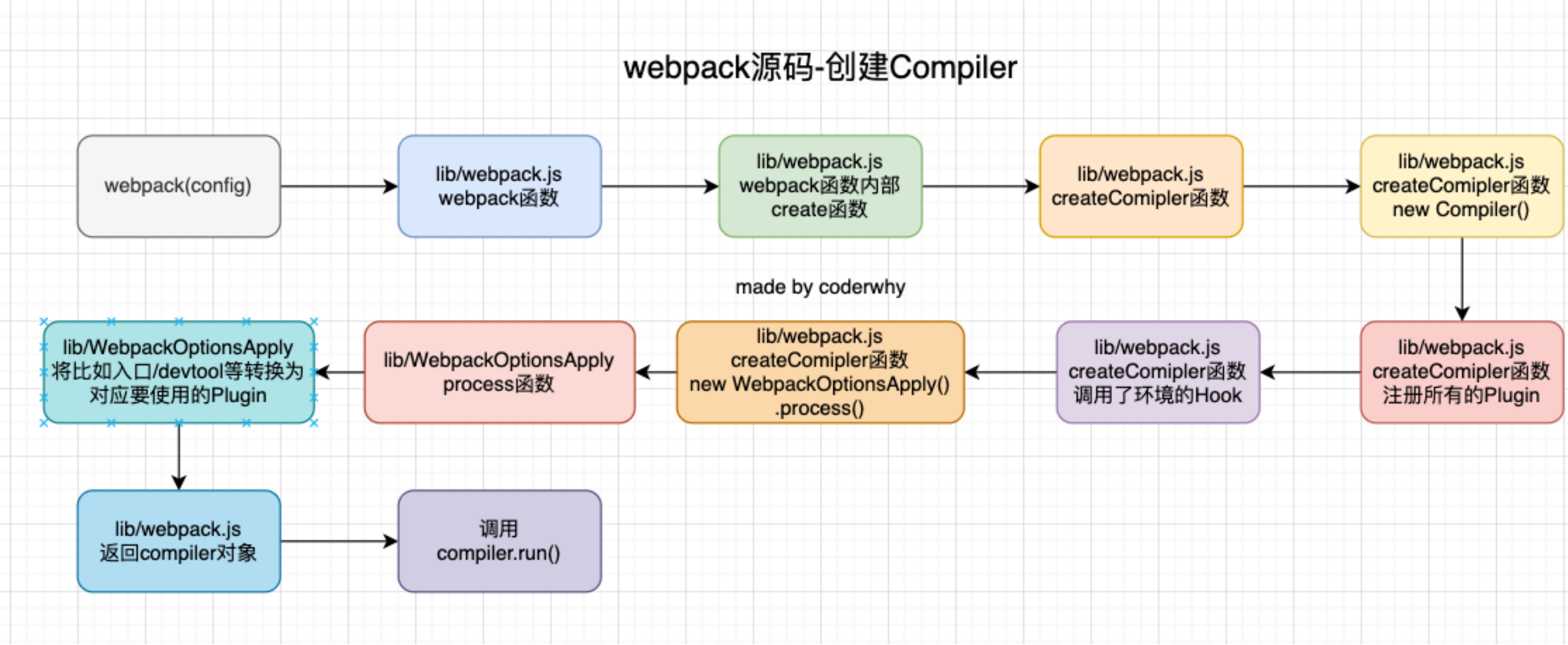
Webpack的启动流程



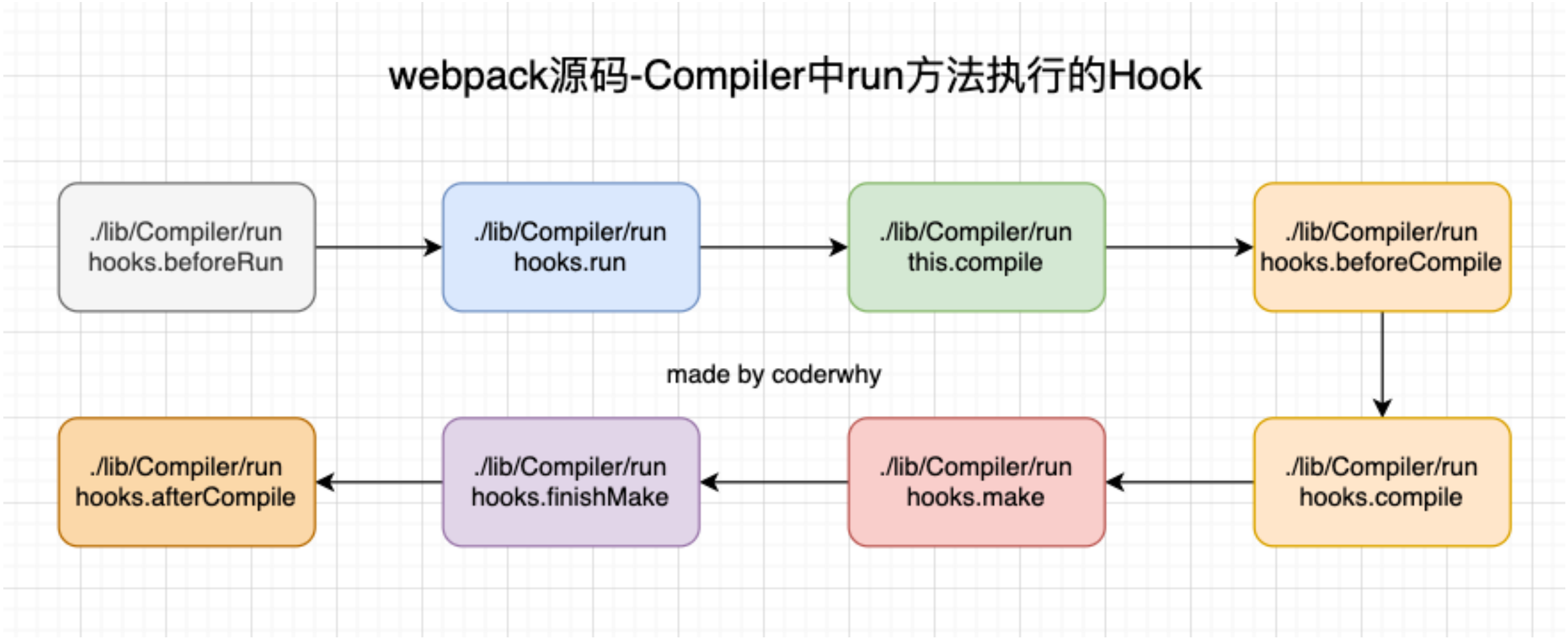
Webpack源码阅读

- 第一步：下载webpack的源码
  - https://github.com/webpack/webpack
- 第二步：安装项目相关的依赖
  - npm install
- 第三步：编写自己的源代码
  - 这里我创建了一个 why 文件夹，里面存放了一些代码
- 第四步：编写webpack的配置文件
  - webpack.config.js
- 第五步：编写启动的文件build.js

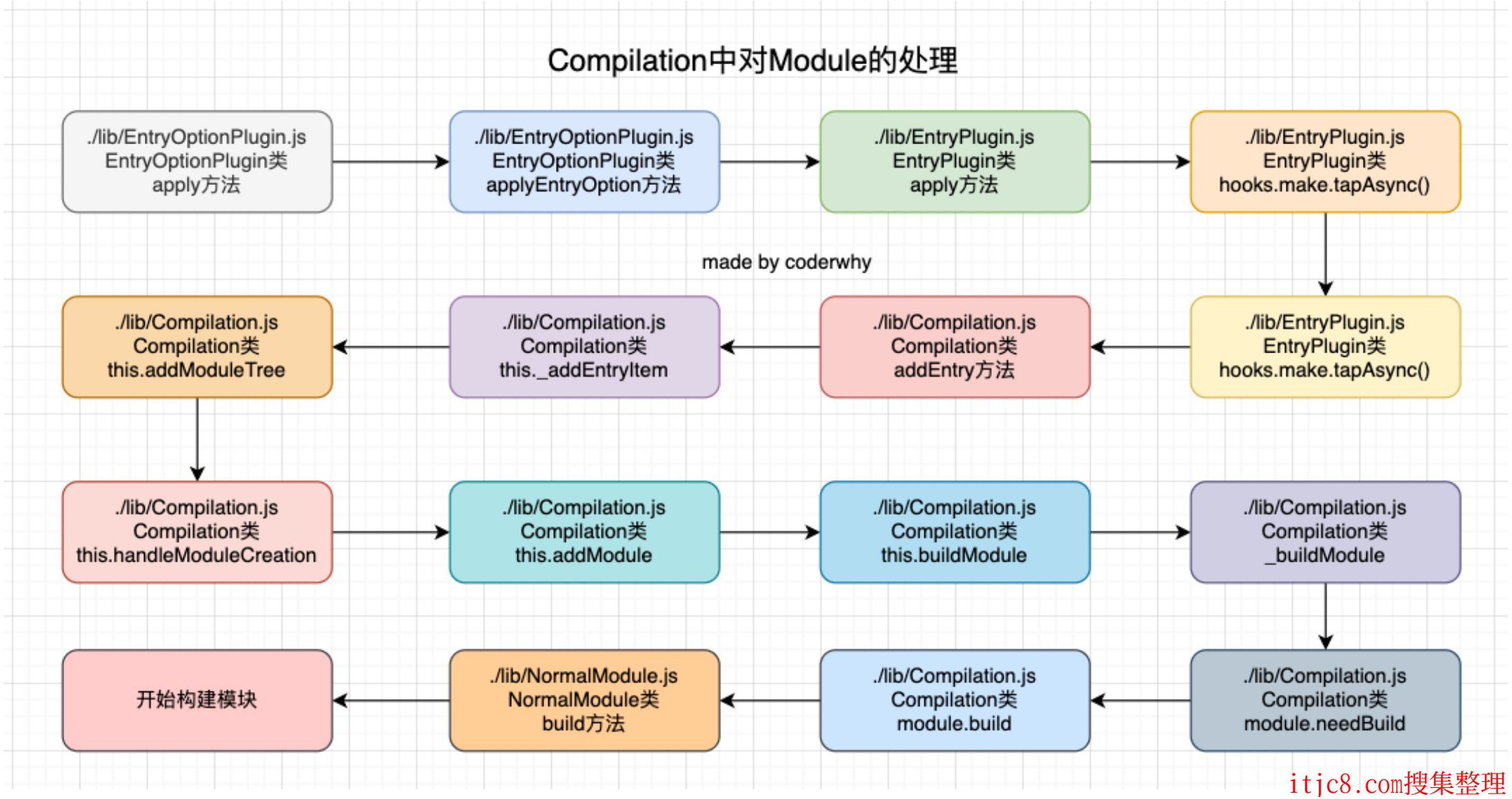
创建Compiler



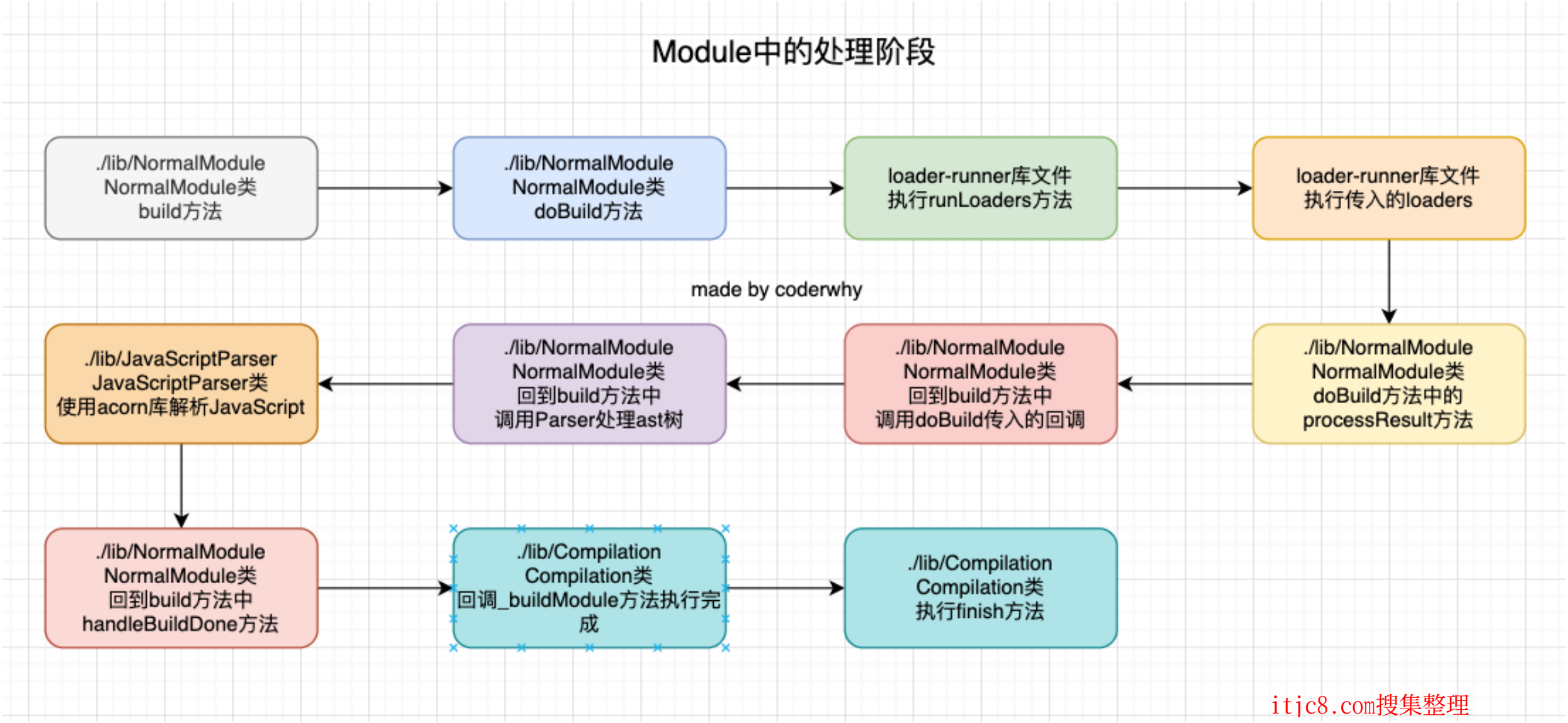
Compiler中run方法执行的Hook



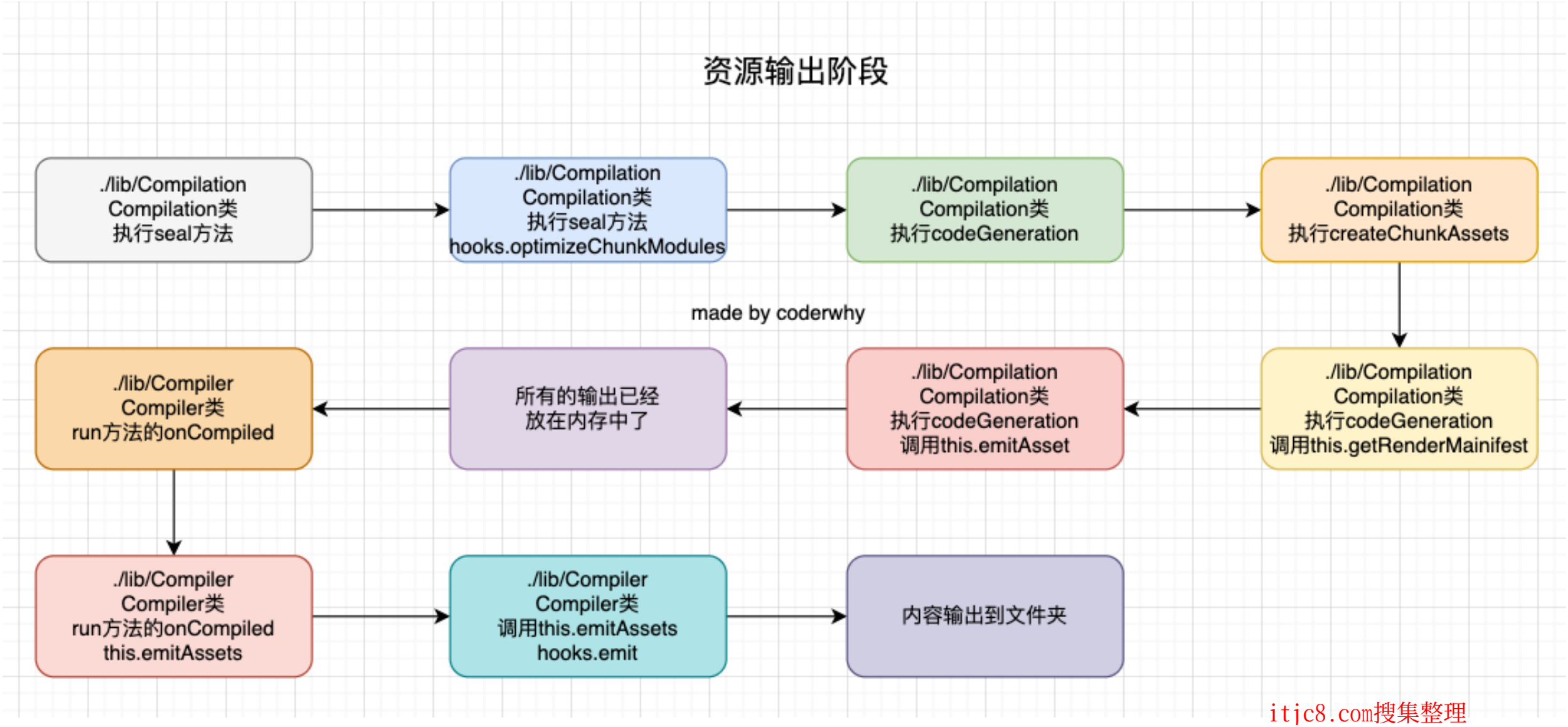
Compilation对Module的处理



module的build阶段



输出asset阶段



Compiler和Compilation的区别

- // Compiler和Compilation的区别
- // 在webpack构建的之初就会创建的一个对象, 并且在webpack的整个生命周期都会存在(before - run - beforeCompiler - compile - make - finishMake - afterCompiler - done)
- // 只要是做webpack的编译, 都会先创建一个Compiler
-
  // Compilation是到准备编译模块(比如main.js), 才会创建Compilation对象
- // 主要是存在于 compile - make 阶段主要使用的对象
-
  // watch -> 源代码发生改变就需要重新编译模块
- // Compiler可以继续使用(如果我修改webpack的配置, 那么需要重新执行run run build)
- // Compilation需要创建一个新的Compilation对象
-