

## 怎么调试 Webpack ?

更新时间：2019-06-25 17:21:36



“

宝剑锋从磨砺出，梅花香自苦寒来。

——佚名

”

在深入理解 Webpack 内核原理之前，我们先来学习下研究方法，即怎么调试 Webpack。我们都知道 Webpack 实际是 Node.js 编写的，执行的时候我们使用 `webpack-cli` 命令进行执行，所以我们可以利用 Node.js 的调试方法来调试 Webpack。

在这里我们不是使用 `console.log` 的方式来调试代码，而是使用断点的方式来调试 Webpack，配合 VSCode IDE 编辑器的调试面板来实现 Webpack 的 Debug。

VSCode 是一个基于 Electron 编写的 IDE，Electron 是基于 Chromium 内置 Node.js 的一套桌面应用解决方案，通过它可以用 JavaScript、HTML 和 CSS 这些前端技术栈来创建桌面应用，并且 Electron 创建出来的桌面应用是支持跨端的！VSCode 和 Electron 都是开源的，感兴趣的可以去 Github 查看源码。本篇文章主要来介绍基于 VSCode 的调试面板来调试 Webpack，学会这个其实可以扩展到调试任意 Node.js 项目代码。

### 前期准备

首先是先下载并安装 VSCode。我电脑是 Mac 系统，所以以 Mac 版本的 VSCode 为教程来介绍，Windows 版本并没有多少差异性；

第二步是准备 Node.js 开发环境。我这里使用的是 10.15.3 LTS 版本的 Node.js（截稿最新版本是 Node.js 12，LTS 版本是 10.15.3），但是调试技巧都是通用的；

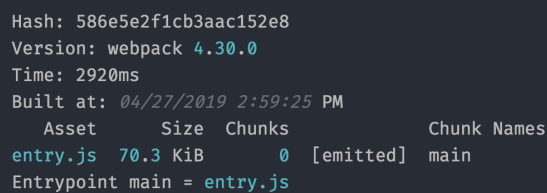
第三步是准备好 Webpack 和 Webpack-cli 工程项目。我们按照下面顺序来创建一个调试项目：

1. 创建一个 `webpack-debug` 的目录并且进入目录，Mac 命令行如下：`mkdir webpack-debug && cd $_`；
2. 创建 `package.json` 初始化 NPM 环境：`npm init -y`；

3. 安装开发依赖: `npm install -D webpack webpack-cli`;
4. 安装项目依赖, 这里使用了 `lodash` 这个工具库: `npm install -S lodash`;
5. 创建项目 Webpack 入口文件 `src/index.js` 和配置文件 `webpack.config.js`, 内容如下:

```
// src/index.js
import _ from 'lodash';
console.log(_.isArray(1));
// webpack.config.js
module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'entry'
  }
};
```

这时候我们执行 `npm webpack`, 执行成功:



```
Hash: 586e5e2f1cb3aac152e8
Version: webpack 4.30.0
Time: 2920ms
Built at: 04/27/2019 2:59:25 PM
    Asset      Size  Chunks             Chunk Names
  entry.js  70.3 KiB       0  [emitted]  main
Entrypoint main = entry.js
```

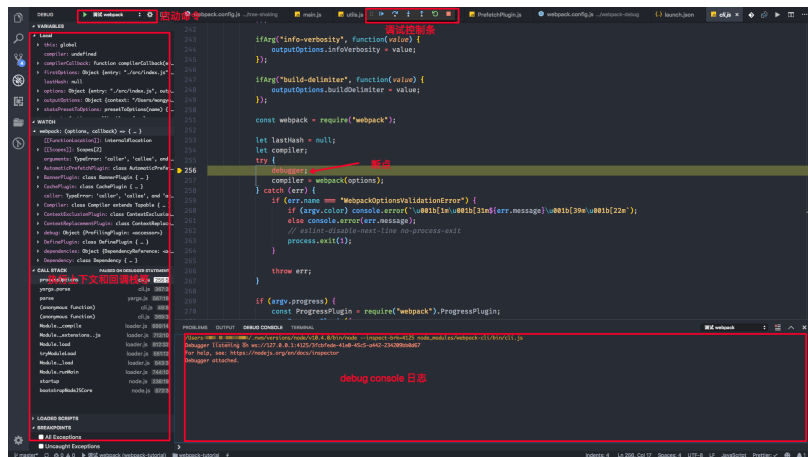
下面来说下 `npm webpack` 做了什么事情。之前介绍下, `npm` 实际执行的是对应模块的 `bin` 命令, `bin` 命令是对应的 `NPM` 模块在模块安装 (`npm install`) 的时候通过 `package.json` 的 `bin` 字段指定的, `webpack` 这个命令实际是通过 `webpack-cli` 来注入的。我们来看下 `webpack-cli` 的 `package.json`:

```
// webpack-cli package.json
{
  // ...
  "bin": {
    "webpack-cli": "./bin/cli.js"
  },
  "main": "./bin/cli.js",
  "engines": {
    "node": ">=6.11.5"
  }
  // ...
}
```

所以实际我们执行 `npm webpack`, 跟执行 `node ./node_modules/webpack-cli/bin/cli.js` 效果是一样的, 知道这个我们就明白原来 `Webpack` 命令实际执行的就是 `webpack-cli` 项目中的 `bin/cli.js` 呢!

## 使用 VSCode 的调试功能

了解了 `webpack` 执行命令的原理之后, 我们再来看下 `VSCode` 的调试功能怎么使用, 下面是一张调试面板的截图:



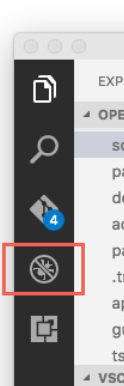
在 VSCode 中开启调试很简单，只需要下面步骤：

打开需要调试的文件，然后在需要的地方设置添加 **debugger** 关键字设置断点；

按下 **F5** 快捷键，这时候会让选择调试项目的类型，选择后会生成一个 **launch.json** 文件；

修改 **launch.json** 文件的相关内容，主要是 **name**、**program** 和 **cwd** 三个字段，改成项目对应的配置即可。这里使用 **`\${workspaceFolder}`** 这个变量，代表 VSCode 的工作路径，即当前打开项目的根目录；

点击 VSCode 界面左侧长得类似虫子的按钮进入 **debug** 面板界面，点击上图中左上角的开始三角按钮，启动调试；



5. 这时候就可以使用调试控制条进行调试了。左侧会显示执行上下文、变量、断点、调用栈等多个信息，跟 Chrome 的 DevTools 调试非常类似。

调试控制条如下图所示，共有 6 个按钮，从左到右依次为下面列表及其对应的快捷键（Mac 系统）：



- 继续/暂停： **F5**
- 单步跳过： **F10**
- 单步调试： **F11**
- 单步跳出： **⇧F11**
- 重新调试： **⇧⇧F5**
- 停止调试： **⇧F5**

## 使用 VSCode 调试 Webpack

下面我们来动手实践下如何用 VSCode 调试 Webpack，整个调试的流程最关键的步骤是两步：

1. 寻找 `cli.js` 中合适的断点位置；
2. 配置 VSCode 的 `launch.json`。

寻找 `cli.js` 中合适的断点位置

首先我们知道执行 `webpack` 执行的文件是 `./node_modules/webpack-cli/bin/cli.js`，我们需要了解下这个文件的大概执行流程，并且找出合适的断点位置。

下面我从头来分析整个 `cli.js` 源码寻找断点位置。这个过程我会比较详细地来讲解，我希望大家能够在我操作的过程中，学习到如何有侧重点的阅读开源的代码，既能了解整体的实现方案又能够找到合适的断点位置。

先看下 `cli.js` 的文件大框架：

```
const {NON_COMPILATION_ARGS} = require('./utils/constants');

(function() {
  const importLocal = require('import-local');
  // Prefer the local installation of webpack-cli
  if (importLocal(__filename)) {
    return;
  }

  require('v8-compile-cache');

  const ErrorHelpers = require('./utils/errorHelpers');

  const NON_COMPILATION_CMD = process.argv.find(arg => {
    if (arg === 'serve') {
      global.process.argv = global.process.argv.filter(a => a !== 'serve');
      process.argv = global.process.argv;
    }
    return NON_COMPILATION_ARGS.find(a => a === arg);
  });

  if (NON_COMPILATION_CMD) {
    return require('./utils/prompt-command')(NON_COMPILATION_CMD, ...process.argv);
  }

  const yargs = require('yargs').usage(`webpack-cli ${require('../package.json').version}`

  Usage: webpack-cli [options]
    webpack-cli [options] --entry <entry> --output <output>
    webpack-cli [options] <entries...> --output <output>
    webpack-cli <command> [options]

  For more information, see https://webpack.js.org/api/cli/.`);

  require('./config/config-yargs')(yargs);

  yargs.parse(process.argv.slice(2), (err, argv, output) => {
    // 这里才是整个 bin 命令的主体
    // ...
  });
})();
```

整个文件中，主要是使用 `Yargs` 这个包来做 `bin` 命令的选项解析器。通过 `Yargs` 可以解析出执行的是什么命令，从而调用对应的方法做对应的事情。整个文件的主体在：

```
yargs.parse(process.argv.slice(2), (err, argv, output) => {
  // 这里才是整个 bin 命令的主体
  // ...
});
```

我们来看下 `yargs.parse` 解析完 `bin` 命令参数之后在内部做的事情：

```
yargs.parse(process.argv.slice(2), (err, argv, output) => {
  Error.stackTraceLimit = 30;

  // 解析失败需要输出内容时，输出内容
  if (err && output) {
    console.error(output);
    process.exitCode = 1;
    return;
  }

  // help 和 version 信息
  if (output) {
    console.log(output);
    return;
  }

  if (argv.verbose) {
    argv['display'] = 'verbose';
  }

  let options;
  try {
    // 根据命令行传入的 argv 参数，生成 processOptions 的参数对象
    options = require('./utils/convert-argv')(argv);
  } catch (err) {
    // 解析失败，输出错误信息
    if (err.name !== 'ValidationError') {
      throw err;
    }

    const stack = ErrorHelpers.cleanUpWebpackOptions(err.stack, err.message);
    const message = err.message + '\n' + stack;

    if (argv.color) {
      console.error(`\u001b[1m\u001b[31m${message}\u001b[39m\u001b[22m`);
    } else {
      console.error(message);
    }

    process.exitCode = 1;
    return;
  }

  const stdout = argv.silent
    ? {
        write: () => {}
      } // eslint-disable-line
    : process.stdout;

  function ifArg(name, fn, init) {
    // 处理数组型的参数执行命令
    if (Array.isArray(argv[name])) {
      if (init) init();
      argv[name].forEach(fn);
    } else if (typeof argv[name] !== 'undefined') {
      if (init) init();
      fn(argv[name], -1);
    }
  }

  function processOptions(options) {
    // ... 暂时忽略
  }
  processOptions(options);
});
```

通过上面的代码，我们发现 `yargs.parse` 内部一开始主要做了一些参数解析和错误处理的事情，然后重要的是根据命令行传入的 `argv` 参数，生成 `processOptions` 的参数对象（`options`）。这时候整个内部重点变成了 `processOptions(options)` 的执行。下面继续看 `processOptions` 的函数内部实现：

我们会发现，`processOptions` 函数的前面，基本都是根据传入的 `bin` 命令参数在处理 `outputOptions`。`outputOptions` 是跟输出 Webpack 编译结果相关的一个参数对象，比如会显示 Webpack 执行中的错误信息，生成打包报告等，所以不是我们主要关注的。我们关注的是 Webpack 打包过程。所以类似下面的代码我们不需要细研究：

```
ifArg('build-delimiter', function(value) {
  outputOptions.buildDelimiter = value;
});
```

最后我们发现了 `const webpack = require('webpack');` 关键字出现了，这时候整个 `processOptions` 的核心代码出现了，即下面的内容：

```
const webpack = require('webpack');

let lastHash = null;
let compiler;
try {
  compiler = webpack(options);
} catch (err) {
  if (err.name === 'WebpackOptionsValidationError') {
    if (argv.color) console.error(`\u001b[1m\u001b[31m${err.message}\u001b[39m\u001b[22m`);
    else console.error(err.message);
    // eslint-disable-next-line no-process-exit
    process.exit(1);
  }

  throw err;
}
```

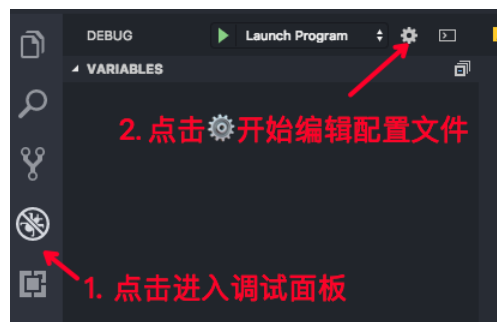
在上面代码中，`cli.js` 导入了 `webpack` 模块依赖，然后开始执行 `webpack(options)`，这时候正式进入 `webpack` 模块的打包流程。我们可以在 `webpack(options)` 之前添加一个 `debugger` 断点，这样调试代码的时候，执行到这里就暂停，等待调试。

**Tips:** 整个源码阅读流程可以大概总结为：

1. 首先折叠不用的分支和逻辑，只看文件大的框架代码；
2. 寻找执行的关键路径，根据变量名称猜测变量意图，重点变量通过仔细阅读对应实现源码来验证想法；
3. 根据关键路径顺藤摸瓜，理清整个执行过程；
4. 了解了整个执行过程之后，进入下一个阶段阅读或者做对应的事情（比如设置断点）继续研究。

修改 `launch.json` 开始调试

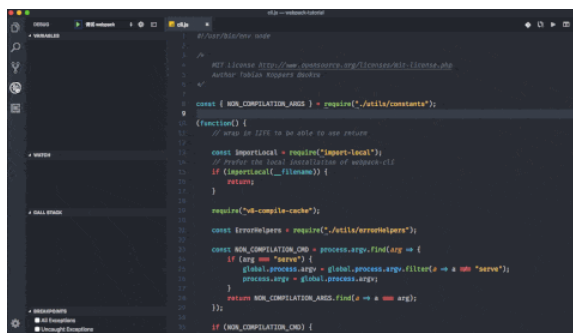
设置完断点之后，这时候在 `cli.js` 页面按 `F5` 快捷键，选择 `Node.js` 环境，这时候会进入编辑 `launch.json`。如果直接进入了调试模式，那么需要按照下图方式手动打开 `launch.json` 编辑界面：



这时候，我们根据项目的实际情况修改 `name`、`cwd`、`program` 配置。如果我们用 `VSCode` 打开的是项目之前创建的 `webpack-debug` 文件夹，那么 `${workspaceFolder}` 对应的就是 `webpack-debug` 的完整路径，配置内容如下：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      // 起个名字
      "name": "调试 webpack",
      // 工作路径
      "cwd": "${workspaceFolder}",
      // 被调试文件的路径，以${workspaceFolder}开头
      "program": "${workspaceFolder}/node_modules/webpack-cli/bin/cli.js"
    }
  ]
}
```

经过上面配置，我们保存 `launch.json` 之后，就可以开始调试了，整个过程可以看下下面的动图：



## 总结

本小节文章主要介绍了怎么使用 `VSCode` 的 `debug` 面板来调试 `Webpack`。其实这种方式不仅仅用于 `Webpack` 调试，也是 `Node.js` 的一种比较轻量级的调试方式。在讲解调试方法的过程中，带领大家阅读了 `webpack-cli` 的 `cli.js` 源码，通过这个过程希望大家可以掌握一种阅读开源的方法，解决面对一个开源项目源码无从下手的窘境。

本小节相关面试题：

1. 怎么 debug 一个 `Node.js` 项目；
2. 平时使用什么 IDE，如何调试代码？

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示



目前暂无任何讨论