Webpack 5.0

更新时间：2019-07-19 12:24:04

> 先相信你自己，然后别人才会相信你。
>
> ——屠格涅夫

Webpack 5.0 从 2018 年年底就开始发布 alpha 版本了，截稿时间，已经过去大半年了还没有发布正式版本。上次 6 月份在 GMTC 大会上 Webpack 的核心开发者 Sean Larkin 说 Webpack 5.0 还需要几个月的时间完成集中的测试，总体来说 Webpack 5.0 离我们不远了。今天的文章来自于 Webpack 团队的 Dennis Gaebel 《New features in Webpack 5》的英文版翻译（翻译参考《(Webpack 5 中的新特性)[https://zhuanlan.zhihu.com/p/56796027]》），要是说 Webpack 5.0 的新特性，没有人比作者自己更了解了！下面开始原文的翻译，其中增加本人对 Webpack 5.0 的一些认识和注解。

你可能已经在使用 Webpack 来打包前端资源，Webpack 即将发布新的 V5 版本。在本文中，我将分享 Webpack 5 的一些新特性，以及在日常工作中继续使用它时应该注意的事项。

该版本新增了大量的新功能，这里简单介绍其中被 Webpack Core Team 提到的核心功能。

预期

在写这篇文章的时候，V5 版本仍然处于早期阶段，可能仍然有问题。作为一个 major 版本，其中有一些 breaking changes，可能会导致一些常见功能或者配置不能做正常工作。Webpack 尝试在可能的情况下提供兼容层，有些更改使其非常难处理，特别是会增加额外的运行时代码。 如果有插件不工作了，可以在这里报告: webpack 5 alpha feedback · Issue #8537 · webpack/webpack 。完整的更新日志可以查看这里：webpack/changelog-v5。值得注意的是：**Webpack 5 将不支持 Node.js 8 以前的版本**（Webpack 4 最低 Node.js 版本为 6）。

Webpack V5 版本主要集中在几个关键组（功）件（能）上：

- 使用持久化缓存提高构建性能；

- 使用更好的算法和默认值改进长期缓存（long-term caching）；
- 清理内部结构而不引入任何破坏性的变化；
- 引入一些 breaking changes，以便尽可能长的使用 V5 版本。

同样，这里是完整的更新日志，但一定要确保在配置更改之前进行阅读，以保持最新。

> 注解：Webpack 的编译速度和配置项之多一直被人诟病，所以 Webpack 5 主要是在构建速度和零配置项上面做了很多改进，并且清理了一些废弃的配置项写法。

要测试 V5 版本，你可以使用以下命令安装它：

```
npm install webpack@next --save-dev
```

> Tips：使用 Webpack V5 则需要 Node.js 8 以上版本！

这个命令引用了最新的 alpha 版本，但是你也可以使用以下命令通过 Webpack 的仓库中的 tag 使用 V5 开发中的版本:

```
npm install webpack@v5.0.0-alpha.18 --save-dev
```

> Tips：当前最新 alpha 版本是 alpha.18 。

如果你使用 Webpack V4 或更高版本，你还需要安装 CLI：

```
npm install webpack-cli --save-dev
```

> 注解：Webpack-cli 只是一个 CLI 工具，实际内部使用的是 Webpack，而且本身 Webpack-cli 没有锁定 Webpack 版本号，所以 Webpack-cli 支持 Webpack V4 而且也支持 V5。

废弃项目

V4 中已弃用的所有项目已在 V5 中删除。 当迁移到 V5 时，请确保 Webpack 4 版本不会打印弃用警告。 如果您遇到错误问题，请尝试省略 stats 选项或不使用预设。 尽管如此，事情仍然处于发布前的阶段，所以最好通过 GitHub 向 Webpack 团队咨询。

也有一些 V4 中没有被警告 deprecation 的变更，比如 IgnorrePlugin 和 BannerPlugin，现在必须传递一个 options 对象。 下面是一个 IgnorrePlugin 的示例，当前的文档似乎没有提到这一点:

```
new webpack.IgnorePlugin({ resourceRegExp: regex })
```

相关阅读

- banner plugin
- ignore plugin

默认 Node.js 的 Polyfill 移除

过去，Webpack 的目标是允许在浏览器中运行大多数 Node.js 模块，但是模块版本发生了变化，许多模块的使用现在都是专门为前端而编写的。 在 Webpack V4 附带了大多数 Node.js 核心模块的 Polyfills，一旦模块使用了任何核心模块，这些模块就会自动应用。

反过来，这又将这些大的 Polyfill 添加到最后的 Bundle 中，但通常是不必要的。 V5 中的尝试是自动停止引入 Node.js 的这些 Polyfill 代码，并侧重于前端兼容的模块。当迁移到 V5 时，最好尽可能使用前端兼容的模块，并尽可能手动添加核心模块的 Polyfill（错误消息可以帮助指导您）。 对于核心团队的反馈，我们表示感谢 / 鼓励，因为这个更改可能会也可能不会进入最终的 V5 版本。

> **注解**：Webpack V5 版本不再自动引入 Node.js 模块的 Polyfill，如果要在 Web 页面使用 Node.js 的模块，则需要手动自己添加 Polyfill，Node.js 的 Polyfill 可以参考这个列表。对于这个修改不是最终结论，大家可以在 Webpack 的 Github 上面讨论并且提供反馈，Webpack 核心团队（Core Team）会根据反馈来做是否将该修改纳入 Webpack V5 正式版的参考。
> **点评**：这点是很有意思的，明确了我们编写前端页面没有必要一股脑的使用 Node.js 的这类后端模块 Polyfill，但是由于 Webpack 之前内置了 Node.js Polyfill 太方便了，导致了使用者都忘记在 Web 页面中不能使用 Node.js 的模块了，升级成 V5 之后，老项目需要手动引入 Node.js 的 Polyfill 的确有点麻烦，所以这个特性还需要讨论，有诉求的可以直接跟 Webpack 团队反馈。个人觉得这个方案思路是正确的，Web 前端开发者更加专注自己的模块，对自己的代码更加了解，稀里糊涂的写代码没做任何配置 Webpack 打包出来却没有错，这样下去总会留下一些隐性坑。

确定性 ChunkId 和 ModuleId

为了增强长期缓存（long-term caching），在 Webpack V5 版本增加了新的算法，并在**生产模式下使用以下配置开启**：

```
chunkIds: "deterministic",
moduleIds: "deterministic"
```

这些算法以确定性的方式为 module 和 chunk 分配非常短（3 或 4 个字符）的数字 id。 这是 Bundle 大小和长期缓存之间的权衡。 从 V4 迁移时，最好使用 `chunkIds` 和 `moduleIds` 的默认值。 你也可以从配置文件中选择旧的默认设置：

```
chunkIds: "size",
moduleIds: "size"
```

上面的配置将生成较小的 Bundle，但由于缓存的原因，它们更容易失效。

> 注解：在 Webpack 中使用类似 `import().then` 的语法异步按需引入一个模块的时候，在 Webpack4 中会被打包出来类似 `0.js` 或者 `1.js` 这种按照数字编号的 chunkId，这里有个问题就是加入我们使用 `import()` 动态引入多个模块，那么会按照编号顺序加载这些模块，加入我们添加的模块减少了，那么这些编号还会变化，这就导致了我们打包出来的 entry 文件内容发生了变化，例如之前加载的 `0.js` 对应的是 `src/async0.js`， `1.js` 对应的是 `src/async1.js`，而现在移除了 `src/async0.js`，则 `0.js` 对应的是 `src/async1.js` 文件了，这样就导致虽然 `src/async1.js` 内容没有变化，而 http 请求的时候 url 发生了变化，所以不得不从服务端重新拉取最新 url 的代码，从而导致缓存失效。当然在 Webpack V4 中我们可以通过 `Magic Comment` 的方式来给每个异步引入的模块添加固定的 `webpackChunkName`，但是这些在 Webpack V5 中不需要配置了，类似的 chunk 都会被**固定**的分配非常短的数字 id，而这些 id 在编译过程中是不会因为移除文件而发生变化的。不过虽然这样，数字 id 的 chunk 名称对于某些小伙伴来说还是不能忍，那么还是使用 magic comments 给每个引入的模块加上固定的名字吧。

相关阅读

- https://webpack.js.org/configuration/optimization/#optimization-moduleids
- https://webpack.js.org/configuration/optimization/#optimization-chunkids

chunkId 的命名

在默认情况下，为了方便在**开发模式**进行调试开发，Webpack 设计了一种新的 ChunkId 命名算法，ModuleId 由对应的路径来决定，而 ChunkId 则由 chunk 的内容而决定，在这种方式下，我们不需要使用 Magic Comments 的方式 `import(/* webpackChunkName: "name" */ "module")` 来手动给页面增加固定的名字。

`import(/* webpackChunkName: "name" */ "module")` 可以用于调试，但是如果你想控制**生产环境**的文件名，这一行也是有意义的。 在产品中使用 `chunkIds:"named"` 是可能的，只是确保不会意外地暴露有关模块名称的敏感信息。当从 v4 进行迁移时，你可能会发现不喜欢在**开发模式**下改变文件名。考虑到这一点，你可以传递下面的一行，以便从配置文件中使用旧的数值模式。

```
chunkIds: "natural"
```

相关阅读

- https://webpack.js.org/configuration/optimization/#optimizationchunkids

> 注解：这一段说的 chunkId 的命名就是针对上一段确定下的 chunkId 和 moduleId 而言的。

Compiler 流程

在新的版本中，编译器在使用完毕后应该被关闭，因为它们在进入或退出空闲状态时，拥有这些状态的 hook。 插件可以用这些 hook 来执行不太重要的工作（比如：持久性缓存把缓存慢慢地存储到磁盘上）。同时插件的作者应该预见到某些用户可能会忘记关闭编译器，所以 当编译器关闭所有剩下的工作时应尽快完成。 然后回调将会通知已彻底完成。

当你升级到 V5 时，请确保在完成工作后使用 Node.js API 调用 Compiler.close。

在 V5 新版本中，Compiler 在使用后将被要求关闭（close），因为它们在进入和离开空闲状态时，拥有用于这些状态的 Hook。 插件可以使用这些 Hook 来做一些不重要的工作（例如，持久性缓存把缓存慢慢地存储到磁盘上）。当 Compiler 关闭所有剩余的工作应尽快完成，然后一个回调将会通知 Compiler 的工作已经彻底完成。

插件和它们各自的作者应该预料到一些用户可能会忘记关闭编译器，因此所有的工作最终应该在空闲时完成。 在工作进行期间，还应防止流程退出。 当传递回调时， webpack() facade 自动调用关闭。 当你升级到 V5 时，请确保在完成工作后使用 Node.js API 调用 Compiler.close 。

> 注解：Compiler 的流程增加了一个close的 hook，通过这个 hook 回调，我们可以做一些收尾（不重要）的工作。

相关阅读

- https://webpack.js.org/api/compiler-hooks

SplitChunks 和 module 大小

在新版本中，module（模块）现在能够以更好的方式拆分体积，而不是显示单个数字和不同类型的体积大小。 SplitChunksPlugin 现在知道如何处理这些不同的体积大小，并将它们用于 minSize 和 maxSize 。 默认情况下，只能处理 javascript 的大小，但是你现在可以传递多个值来管理它们:

```
minSize: {
  javascript: 30000,
  style: 50000,
}
```

相关阅读

- https://webpack.js.org/plugins/split-chunks-plugin

> 注解：在 Webpack V4 版本，我们可以通过splitChunks的配置来制定每个被 split 出来的 chunk 大小，在 V5 中可以根据类型来制定拆包的大小了。

例如 V4：

```
module.exports = {
  optimization: {
    splitChunks: {
      cacheGroups: {
        commons: {
          chunks: 'all',
          name: 'commons',
          minChunks: 1,
          // 体积大小数值
          minSize: '数值',
          maxSize: '数值'
        }
      }
    }
  }
};
```

现在 V5：

```
module.exports = {
  optimization: {
    splitChunks: {
      cacheGroups: {
        commons: {
          chunks: 'all',
          name: 'commons'
        }
      },
      //最小的文件大小 超过之后将不予打包
      minSize: {
        javascript: 0,
        style: 0
      },
      //最大的文件 超过之后继续拆分
      maxSize: {
        javascript: 1, //故意写小的效果更明显
        style: 3000
      }
    }
  }
};
```

持久化缓存

在 V5 版本中，使用者可以找到一个实验性的文件系统缓存，这个缓存是通过 Webpack 配置文件中的下面一行启用下面的配置而打开的：

```
cache: {
  type: 'filesystem';
}
```

目前为止，该功能的核心功能已经达到 Ready 状态。在使用它的时候，你必须意识到它的局限性，以避免意想不到的错误。如果你不完全理解这些限制，你最好完全避免使用这个功能，直到你真正感到舒服为止。

你还可以通过自动缓存失效来解析模块源代码和文件系统结构，但是没有针对配置和 loader / plugin / core 更改的自动缓存失效。 如果想手动缓存无效化，可以在 cache.version 配置中进行设置。目前它还没有完全达到 ready 的状态，但是你可以在升级工具依赖（webpack、loader、plugin）或者更改配置的时候通过更新 cache.version 来使一切运行顺利。

如果你希望自动化此操作，最好在 webpack.config.js 中使用 node_modules/.yarn-integrity 的 hash（这块是根据原文的理解，专栏作者做的翻译解释，这里原文的 cache.version 怎么使用没有给出示例，能够查找到的内容是可能跟 cache-loader 的 cacheIdentifier 用法类似）。 然后将它们传递给 cache.version，这很可能就是 Webpack 团队在内部实现它的方式。

当使用持久缓存时，不再需要 cache-loader，同时也不需要对 Babel cacheDirectory 设置为 true 。

相关阅读

- https://webpack.js.org/configuration/other-options/#cache

> **特别注意**：关注 Webpack 5 持久化缓存 cache 的配置项目在网上涉及到的内容较少，这里虽然说有 cache.version 的用法，但是具体使用方式并没有给出示例，这块猜测和 cache-loader 的配置有相关性，从原文来看，这块 Webpack 的内部实现还没有完成，要等 Webpack 5 正式版本发布之后，我们再来看这块究竟怎么使用。

> **注解**：在 Webpack 5 中为了提升 Webpack 的编译速度，引入了持久缓存功能，该功能是将编译的中间产物通过存入文件系统（filesystem，即存入磁盘），再次打包的时候会直接读取中间产物以提升速度。而且持久化缓存不仅仅针对 Webpack 进行优化，对于 Babel 也进行了优化，从而减少配置项，降低 Webpack 缓存的使用门槛。该 feature 灵感来自于该 issue 的讨论，想了解实现思路的可以通过该issue 来查看讨论过程。

配置更改

因为要列出的配置更新太多了，所以你可以通过 v5 changelog 读取所有关于配置更新的信息。

下面是 Sean Larkin 放出的 Webpack 5 修改的 list：

> **注解**：由此可见 Webpack 5 的配置项修改很多，但是当扫描完整个文档之后会发现，其实修改的一些项目基本分为几类：1. 默认配置项修改，已经为我们做好了兼容和最佳实践；2. 增加配置项多数为很少用的高级选项；3. 多数修改围绕本文提到的一些新特性。所以整体来说不用太担心 Webpack 4 迁移到 5 需要大规模的修改webpack.config.js内容。

Webpack 的内部变化

有一些内部变化是插件作者可能需要关注的。 如果你需要查看这些变更，可以通过这个 changelog 来查询内部更改。

最后的一些想法

如果你发现一个混淆或需要进一步帮助的错误，请务必将你的问题在这个 issue中进行反馈。在发布问题之前，可以通过浏览其他开发人员发布的评论来寻找答案。

最后，确保在报告问题之前尝试升级最新的 alpha 版本，因为它可能已经被修复了。 Happy bundling!

专栏作者点评

Webpack 4 推出的时候，很多插件和 loader 都跟不上节奏，社区生态需要一段时间才能适配 Webpack 4。所以由此推测，即使今年 Webpack V5 能够发布正式版本，但是社区建设还需要持续一段时间。好在 Webpack 5 立志于能够稳定，保证 V5 版本可以更长时间的维护，而不是过段时间推出 V6。

另外，一些核心的 Webpack 插件，可能会左右 Webpack 5 的设计，比如 html-webpack-plugin，这些插件已经是 Webapck 生态的重要组成，所以 Webpack 发布正式版之前必然要测试主要插件的功能，防止升级之后一些插件某些功能失效。

总体来说，Webpack 5 还是挺值得期待的，尤其是编译速度的提升。

实战：给 Webpack 项目添加 modern 模式打包

课程总结

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