



rollup.js

标题：Rollup 模块化打包工具

第0章 前置知识

0.1 ES 标准模块化规范-概述

历史上，JavaScript 一直没有模块（module）体系，无法将一个大程序拆分成互相依赖的小文件，再用简单的方法拼装起来。其他语言都有这项功能，比如 Ruby 的 `require`、Python 的 `import`，甚至就连 CSS 都有 `@import`，但是 JavaScript 任何这方面的支持都没有，这对开发大型的、复杂的项目形成了巨大障碍。

在 ES6 之前，社区制定了一些模块加载方案，最主要的有 CommonJS 和 AMD 两种。前者用于服务器，后者用于浏览器。ES6 在语言标准的层面上，实现了模块功能，而且实现得相当简单，完全可以取代 CommonJS 和 AMD 规范，成为浏览器和服务器通用的模块解决方案。

ES6 模块的设计思想是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonJS 和 AMD 模块，都只能在运行时确定这些东西。比如，CommonJS 模块就是对象，输入时必须查找对象属性。

一个模块就是一个独立的文件。该文件内部的所有变量，外部无法获取。

如果你希望外部能够读取模块内部的某个变量，就必须使用 `export` 关键字导出该变量。想要是要其他模块导出的内容，就必须使用 `import` 导入；

0.2 导入 (Importing)

导入的值不能重新分配，尽管导入的对象和数组可以被修改（导出模块，以及任何其他导入，都将受到该修改的影响）。在这种情况下，它们的行为与 `const` 声明类似。

0.2.1 命名导入 (Named Imports)

从源模块导入其原始名称的特定项目。

```
1 | import { something } from './module.js';
```

从源模块导入特定项，并在导入时指定自定义名称。

```
1 import { something as somethingElse } from './module.js';
```

0.2.2 命名空间导入 (Namespace Imports)

将源模块中的所有内容作为对象导入，将所有源模块的命名导出公开为属性和方法。默认导出被排除在此对象之外。

```
1 import * as module from './module.js'
```

上面的“something”的例子将被附加到作为属性的导入对象上。“module.something”。

0.2.3 默认导入 (Default Import)

导入源文件的 **默认导出**

```
1 import something from './module.js';
```

0.2.4 空的导入 (Empty Import)

加载模块代码，但不要创建任何新对象。

```
1 import './module.js';
```

这对于polyfills是有用的，或者当导入的代码的主要目的是与原型有关的时候。

0.3 导出 (Exporting)

0.3.1 命名导出 (Named exports)

导出具体声明的值：

```
1 var something = true;  
2 export { something };
```

在导出时重命名：

```
1 export { something as somethingElse };
```

声明后立即导出：

```
1 // 这可以与 `var`, `let`, `const`, `class`, and `function` 配合使用  
2 export var something = true;
```

0.3.2 默认导出(Default Export)

导出一个值作为源模块的默认导出：

```
1 | export default something;
```

仅当源模块只有一个导出时，才建议使用此做法。

将默认和命名导出组合在同一模块中是不好的做法，尽管它是规范允许的。

第1章 Rollup 入门

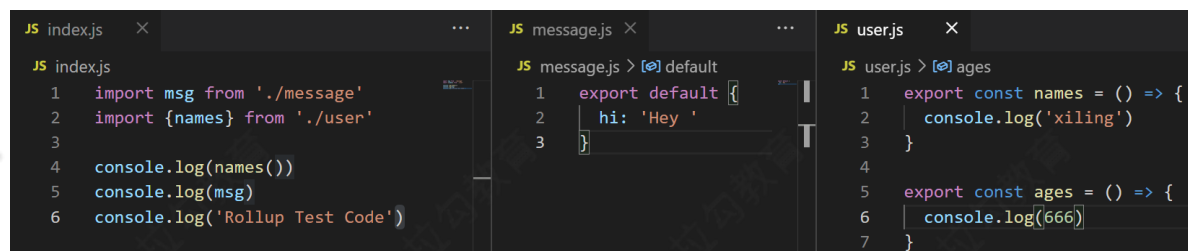
1.1 概述(Overview)

Rollup 是一个 JavaScript 模块打包器，可以将小块代码编译成大块复杂的代码，。Rollup 对代码模块使用新的 ES Module 标准，而不是以前的社区模块化方案，如 CommonJS 和 AMD。ES6 模块已经被现代化浏览器原生实现，2020 年 Vue 作者发布了 Vite，定义为下一代前端开发与构建工具，目的是取代依赖 Webpack 打包的各种构建工具，而 Vite 的模块化工具使用的就是 Rollup；

1.2 快速入门

使用 `npm install --global rollup` 进行安装, 运行 `rollup --help` 可以查看可用的选项和参数。

帮助信息中的提示中: `Usage: rollup [options] <entry file>` 就展示了 Rollup 的基本用法, 我们准备三个文件 `index.js` 为入口主文件, 然后使用 ES6 模块化语法引入 `user.js` 和 `message.js` 文件, 并在 `mian.js` 中使用两个文件导出的数据内容;



```
JS index.js ×
JS index.js
1 import msg from './message'
2 import {names} from './user'
3
4 console.log(names())
5 console.log(msg)
6 console.log('Rollup Test Code')

JS message.js ×
JS message.js > [0] default
1 export default {
2   hi: 'Hey '
3 }

JS user.js ×
JS user.js > [0] ages
1 export const names = () => {
2   console.log('xiling')
3 }
4
5 export const ages = () => {
6   console.log(666)
7 }
```

```
1 // ===== index.js =====
2 import msg from './message'
3 import {names} from './user'
4
5 console.log(names())
6 console.log(msg)
7 console.log('Rollup Test Code')
8
9 // ===== message.js =====
10
11 export default {
12   hi: 'Hey '
```

```

13 }
14
15 // ===== user.js =====
16
17 export const names = () => {
18   console.log('xiling')
19 }
20
21 export const ages = () => {
22   console.log(666)
23 }

```

命令行终端执行: `rollup index.js`

```

1 $ rollup index.js
2
3 index.js → stdout...
4 var msg = {
5   hi: 'Hey '
6 };
7
8 const names = () => {
9   console.log('xiling');
10 };
11
12 console.log(names());
13 console.log(msg);
14 console.log('Rollup Test Code');

```

在命令终端中, 我们能看到最终打包后的编译代码被打印出来了, 我们需要的是将小块的文件代码打包到一个大的文件中, 想要实现效果, 只需要在命令中添加指定的命令参数即可: `rollup index.js --file bundle.js`, 生成 `bundle.js` 文件, 代码如下:

```

1 var msg = {
2   hi: 'Hey '
3 };
4
5 const names = () => {
6   console.log('xiling');
7 };
8
9 console.log(names());
10 console.log(msg);
11 console.log('Rollup Test Code');

```

我们看到, 打包后的代码非常简洁, 就是将需要运行的代码, 按照顺序拼装到一起, 注意, 是只会保留用到的代码, 这就是 Rollup 最早提出的 Tree-shaking 特性, 后来被几乎所有打包工具参考引入;

什么是 Tree-shaking 呢? 基本原理非常简单, Rollup 在打包的过程中, 静态分析代码中的 `import`, 只引入最基本最精简代码, 并将排除任何未实际使用的代码。所以可以生成轻量、快速, 以及低复杂度的 library 和应用程序。

1.3 配置文件

我们一般在命令行中使用 Rollup。你也可以提供一份配置文件（可要可不要）来简化命令行操作，同时还能启用 Rollup 的高级特性，配置文件是一个ES6模块，它对外暴露一个对象，这个对象包含了一些 Rollup需要的一些选项。通常，我们把这个配置文件叫做 `rollup.config.js`，它通常位于项目的根目录；

注意: Rollup 本身会处理配置文件，所以可以使用 `export default` 语法——代码不会经过 Babel 等类似工具编译，所以只能使用所用 Node.js 版本支持的 ES2015 语法。

配置选项列表: <https://www.rollupjs.com/guide/big-list-of-options>

```
1 // rollup.config.js
2 export default {
3   // 包的入口点 (例如: 你的 main.js 或者 index.js)
4   input: 'index.js',
5   // 出口配置
6   output: {
7     file: 'bundle.js', //打包到那个文件
8     format: 'esm' // 生成包的格式
9   }
10 }
```

输入: input [string]: 'index.js' 这个包的入口点 (例如: 你的 main.js 或者 app.js 或者 index.js)

输出: output [object]: {}

output.file [string]: 'bundle.js' 要写入的文件。如果你想要用的话,生成 sourcemaps也是可以的;

output.format [string]: 'iife' 生成包的格式。包格式选项可以是下面的任意一个:

- `amd` - 异步模块定义, 用于像RequireJS这样的模块加载器
- `cjs` - CommonJS, 适用于 Node 和 Browserify/Webpack
- `esm` - 将软件包保存为 ES 模块文件, 在现代浏览器中可以通过 `<script type=module>` 标签引入
- `iife` - 一个自动执行的功能, 适合作为 `<script>` 标签。(如果要为应用程序创建一个捆绑包, 您可能想要使用它, 因为它会使文件大小变小。)
- `umd` - 通用模块定义, 以 `amd`, `cjs` 和 `iife` 为一体
- `system` - SystemJS 加载器格式

如果你想使用Rollup的配置文件, 记得在命令行里加上 `--config` 或者 `-c`

如果需要不同的配置, 也可以指定与默认 `rollup.config.js` 文件不同的配置文件:

```
1 rollup --config rollup.config.dev.js
2 rollup --config rollup.config.prod.js
```

第2章 使用插件

2.1 认识插件及插件的基本使用

目前为止，我们通过相对路径，将一个入口文件和一个模块创建成了一个简单的 bundle。随着构建更复杂的 bundle，通常需要更大的灵活性——引入 npm 安装的模块、通过 Babel 编译代码、和 JSON 文件打交道等。

为此，我们可以用 插件(plugins) 在打包的关键过程中更改 Rollup 的行为；

Rollup 拥有非常丰富的插件体系，可以去 [插件列表: https://github.com/rollup/awesome](https://github.com/rollup/awesome) 看一下；

这里我们将使用 [rollup-plugin-json](#) 插件，使 Rollup 能够读取 JSON 文件中的数据。

将 rollup-plugin-json 安装为开发依赖，因为代码实际执行时不依赖这个插件——只是在打包时使用：

```
1 npm install rollup-plugin-json -D
```

编辑 `rollup.config.js` 文件，加入 JSON 插件：

```
1 // 直接使用 ES module 导入插件
2 import json from 'rollup-plugin-json';
3
4 export default {
5   input: './src/index.js',
6   output: {
7     file: './dist/bundle.js',
8     format: 'iife',
9   },
10  // 插件导出的是一个函数，直接调用函数，
11  // 注意，是调用的结果直接给到 插件数组中
12  plugins: [ json() ]
13 }
```

更改 index.js 的代码，从 package.json 中读取数据：

```
1 import msg from './message'
2 import { names } from './user'
3
4 import { name, version } from '../package.json';
5 console.log(`项目名称:${name},项目版本: ${version}`)
6
7 console.log(names())
8 console.log(msg)
9 console.log('Rollup Test Code')
10
```

打包过后的代码，只保留了 name 和 version 两个内容，这也就是 tree-shaking 起的作用。

2.2 写一个自己的 Rollup 插件

待更新.....

作者: 西岭老湿

欢迎在 知乎、微信公众号、B站 搜索关注 **西岭老湿**

参考来源:

拉勾教育-大前端高薪训练营

<https://wangdoc.com/es6/module.html>

<https://cn.vitejs.dev/>

<https://www.rollupjs.com/>