参考 rollup [入门指南](rollup.js (rollupjs.org))

全局安装rollup

```
npm install --global rollup
```

创建一个JavaScript文件

```
import foo from "./foo.js"
export default function () {
        console.log(foo);
}
```

```
export default "hello world"
```

现在利用rollup 可以把ESM转换为commonjs格式的代码

```
rollup main.js -f cjs
```

这里的`f`选项是"format"的意思，这个命令会自动将main.js转换为commonjs格式的代码

output

```
'use strict';

var foo = "hello world";

function f () {
        console.log(foo);
}

exports.f = f;
```

如果我们要添加更多配置，这显然是很麻烦的，所以rollup为用户提供了配置文件，每次打包的时候，指定读取该配置文件里面内容即可，默认该配置文件为 "rollup.config.js"

```
// rollup.config.js
export default {
  input: 'src/main.js',
  output: {
    file: 'bundle.js',
    format: 'cjs'
  }
};
```

注意到这里的rollup配置文件使用的ESM导出，也就意味着，你使用的node版本至少应该支持ES Module

配置好文件之后，就不必再次输入繁琐的命令，直接

```
rollup -c
```

也可以通过传递命令行选项的方式，覆写配置文件里面的选项

```
rollup -c -o new-bundle.js
```

这里的`-o`等价于配置文件里面的`file`选项
也可以通过 --config 指定不同的配置文件

```
rollup --config rollup.dev.config
rollup --config rollup.prod.config
```

局部安装rollup

for npm

```
npm install rollup --save-dev
```

for yarn

```
yarn add -D rollup
```

for pnpm

```
pnpm add -D rollup
```

安装完成之后，通常的操作是在`package.json`当中添加一个**build**命令，用以指定**rollup**来进行构建代码

```
{
  "name": "rollup-tutorial",
  "version": "1.0.0",
  "scripts": {
    "build": "rollup -c"
  }
}
```

rollup 插件

> ## rollup 插件机制
>
> **roll**的插件机制比较优秀，故单独提出来讲一下
>
> ## 如何使用插件
>
> 首先用官网的例子来介绍**rollup**的插件是如何使用
>
> 1. 初始化一个项目
>
> ```
> pnpm init
> ```
>
> 2. 局部安装 **rollup**
>
> ```
> pnpm add -D rollup
> ```

3. 安装 `@rollup/plugin-json`

```
pnpm add -D @rollup/plugin-json
```

4. 在`main.js`当中键入以下内容

```
import {version} from "./foo.js"
export default function  () {
        console.log("version" + version)
}
module.exports = main
```

5. 在配置文件`rollup.config.js`当中添加以下配置

```
// rollup.config.js
import json from '@rollup/plugin-json';

export default {
  input: 'src/main.js',
  output: {
    file: 'bundle.js',
    format: 'cjs'
  },
  plugins: [json()]
};
```

然后执行`rollup -c`，输出以下内容

```
'use strict';

var version = "1.0.0";

function main$1 () {
  console.log("version" + version);
}

module.exports = main;

module.exports = main$1;
```

可以看到，rollup 只导入了version字段，其余没有用到的内容都被忽略了,这实际上就是tree-shaking

针对构建产物的插件

有些插件是专门针对已经构建好的代码，比如rollup-plugin-terser这个插件会最小化输出的代码,安装该插件

```
pnpm add rollup-plugin-terser
```

然后编辑rollup配置文件,增加一个最小化的打包构建
这里使用iife格式构建代码，这种格式的代码可以被浏览器用script标签引入，因为我们代码当中有一个导出，这个导出以全局变量的形式供其他代码使用，这里的name就是这个全局变量的名字

```
import {terser} from "rollup-plugin-terser"
export default {
        output: [
                {
                        file: "bundle.min.js",
                        name: "version",
                        format: "iife",
                        plugins: [terser()]
                }
        ]
}
```

执行`rollup -c`然后查看bundle.min.js的内容，可以发现

```
var version=function(){"use strict";return
module.exports=main,function(){console.log("version1.0.0")}}();
```

构建出来的是最小化代码（去除了注释、换行等）

代码分割

rollup 会自动的将代码分割成几个块(chunk),比如说动态加载和多个入口点的情况。还可以使用代码分割的特性来实现懒加载（被导入的模块仅仅只在函数执行之后加载）
把最开始例子里面的静态导入改成动态导入

```
export default function () {
        import ('foo.js').then(({default: foo}) =>
console.log(foo))
}
```

rollup 会使用动态导入来创造一个按需加载的块。为了让 rollup 知道我们生成的这些块文件放置在哪里，需要用`-d`选项传递一个目录

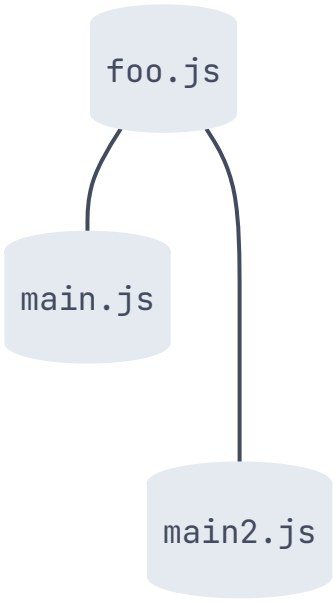
```
rollup src/main.js -f cjs -d dist
```

在dist目录下生成如下两个文件

```
├── foo-d03a9db4.js
└── main.js
```

被分割的出来的chunk文件，都是以chunk-[hash].js的形式命名,这是可以通过`outputchunkfilenames`配置的

多个模块共享同一段代码也会被分割



rollup 不会重复生成两份foo.js

插件概览

rollup 中一个插件是一个对象，拥有一个或者多个属性、构建钩子函数和输出迭代钩子函数。插件遵循rollup的约定（后文细讲）。一个插件应该作为包来发布，这个包应该导出一个能被插件指定选项调用的函数，并且此函数返回一个同样的插件对象
例子
下面这个插件将在不访问文件系统的情况下拦截`virtual-module`的导入。如果想要在浏览器中使用rollup，必须使用这个插件。如下所示，这个插件甚至能替换入口点

```
// rollup-plugin-my-example.js
export default function myExample () {
  return {
    name: 'my-example', // this name will show up in warnings
and errors
    resolveId ( source ) {
      if (source === 'virtual-module') {
        return source; // this signals that rollup should not
ask other plugins or check the file system to find this id
      }
      return null; // other ids should be handled as usually
```

```
    },
    load ( id ) {
      if (id === 'virtual-module') {
        return 'export default "This is virtual!"'; // the
source code for "virtual-module"
      }
      return null; // other ids should be handled as usually
    }
  };
}
```