link: null
title: 珠峰架构师成长计划
description: rollup.config.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=185 sentences=267, words=2496

1. rollup

- 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，例如 library或应用程序

```
import { rollup, watch } from 'rollup';
import inputOptions from './rollup.config.js'
  ; (async function () {

    const bundle = await rollup(inputOptions);

    await bundle.generate(inputOptions.output);

    await bundle.write(inputOptions.output);

    await bundle.close();
  })();
```

rollup.config.js

```
export default {
  input: "./src/index.js",
  output: {
    dir: 'dist',
  }
}
```

package.json

```
{
  "type": "module",
  "scripts": {
    "build": "rollup -c"
  },
}
```

2. rollup插件

- Rollup 插件 (https://rollupjs.org/guide/en/#plugins-overview)是一个具有以下描述的一个或多个 属性、 构建钩子和 输出生成钩子的对象，它遵循我们的约定。插件应该作为一个包分发，该包导出一个可以使用插件特定选项调用的函数并返回这样一个对象
- 插件列表 (https://github.com/rollup/awesome)
- 插件应该有一个清晰的名称，带有 rollup-plugin-prefix
- 在package.json中包含插件关键字
- 插件应该经过测试。我们推荐mocha或ava，它们支持开箱即用的Promise
- 尽可能使用异步方法。
- 编写英文文档
- 如果合适的话，确保你的插件输出正确的 sourcemap
- 如果您的插件使用"虚拟模块"（例如，用于辅助功能），请在模块ID前面加上 \0。这会阻止其他插件尝试处理它
- 插件的名称，用于错误消息和警告
- Type: 字符串
- 为了与构建过程交互，你的插件对象包括"钩子"
- 钩子是在构建的不同阶段调用的函数
- 钩子可以影响构建的运行方式，提供关于构建的信息，或者在构建完成后修改构建
- 有不同种类的钩子
    - async 钩子还可以返回解析为相同类型值的 Promise；否则，钩子将被标记为 sync
    - first 如果有几个插件实现了这个钩子，钩子会按顺序运行，直到钩子返回一个 非null或未定义的值
    - sequential 如果几个插件实现了这个钩子，那么它们都将按照指定的插件顺序运行。如果一个钩子是异步的，那么这种类型的后续钩子将等待当前钩子被解析
    - parallel 如果多个插件实现了这个钩子，那么它们都将按照指定的插件顺序运行。如果一个钩子是异步的，那么这类后续钩子将并行运行，而不是等待当前钩子
- Build Hooks在构建阶段运行，该阶段由 rollup.rollup(inputOptions)触发
- 它们主要负责在 rollup处理输入文件之前定位、提供和转换输入文件
- 构建阶段的第一个钩子是 options，最后一个钩子总是 buildEnd
- 如果出现生成错误，将在此之后调用 closeBundle

plugins\rollup-plugin-build.js

```
import fs from 'fs';

function build() {
  return {
    name: 'build',
    async watchChange(id, change) {
      console.log('watchChange', id, change);
    },
    async closeWatcher() {
      console.log('closeWatcher');
    },
    async options(inputOptions) {
      console.log('options');

    },
    async buildStart(inputOptions) {
      console.log('buildStart');
    },
    async resolveId(source, importer) {
      if (source === 'virtual') {
        console.log('resolveId', source);

        return source;
      }
    },

    async load(id) {
      if (id === 'virtual') {
        console.log('load', id);
        return `export default "virtual"`;
      }
    },
    async shouldTransformCachedModule({ id, code, ast }) {
      console.log('shouldTransformCachedModule');

      return true;
    },
    async transform(code, id) {
      console.log('transform');
    },
    async moduleParsed(moduleInfo) {
      console.log('moduleParsed');
    },
    async resolveDynamicImport(specifier, importer) {
      console.log('resolveDynamicImport', specifier, importer);
    },
    async buildEnd() {
      console.log('buildEnd');
    }
  }
}
export default build;
```

rollup.config.js

```
+import build from './plugins/rollup-plugin-build.js';
export default {
  input: "./src/index.js",
  output: [{
    dir: 'dist',
  }],
  plugins: [
+   build()
  ]
}
```

字段 值 Type (options: InputOptions) => InputOptions null Kind async, sequential Previous Hook 这是构建阶段的第一个钩子 Next Hook buildStart

- 替换或操作传递给 `rollup`的选项对象
- 返回 `null`的话rollup不会替换任何内容
- 如果只需要阅读 `options`，建议使用 `buildStart`钩子，因为在考虑了所有选项钩子的转换后，该钩子可以访问选项
- 这是唯一一个无法访问大多数插件上下文实用程序功能的钩子，因为它是在完全配置汇总之前运行的

字段 值 Type (options: InputOptions) => void Kind async, parallel Previous Hook options Next Hook resolveId并行解析每个入口点

- 每次 `rollup.rollup build`都要调用此钩子
- 当您需要访问传递给rollup的选项时，建议使用这个钩子
- 因为它考虑了所有 `options`钩子的转换，还包含未设置选项的正确默认值

build\plugin-buildStart.js

```
export default function buildStart() {
  return {
    name: 'buildStart',
    buildStart(InputOptions) {
      console.log('buildStart', InputOptions);
    }
  };
}
```

字段 值 Type (source, importer) => string false null Kind async, first Previous Hook

(如果我们正在解析入口点),

（如果我们正在解析导入），或者作为

的后备方案。此外，这个钩子可以在构建阶段通过调用插件钩子触发。

发出一个入口点，或在任何时候通过调用此。

可手动解析id Next Hook 如果解析的

尚未加载，则

，否则

- 定义自定义解析器

- 解析程序可用于定位第三方依赖关系等。这里 source是导入语句中所写的导入对象，即
- 来源就是 "../bar.js"

```
import { foo } from '../bar.js';
```

- importer是导入模块的完全解析id
- 在解析入口点时， importer通常是undefined
- 这里的一个例外是通过 this.emitFile生成的入口点。在这里，您可以提供一个 importer参数
- 对于这些情况， isEntry选项将告诉您，我们是否正在解析用户定义的入口点、发出的块，或者是否为此提供了 isEntry参数。解析上下文函数
- 例如，您可以将其用作为入口点定义自定义代理模块的机制。以下插件将代理所有入口点以注入 polyfill导入
- 返回 null将遵循其他 resolveId函数，最终遵循默认的解析行
- 返回 false信号，表示源应被视为 外部模块，不包括在 bundle中 `

build\plugin-polyfill.js

```
const POLYFILL_ID = '\0polyfill';
const PROXY_SUFFIX = '?inject-polyfill-proxy';

export default function injectPolyfillPlugin() {
  return {
    name: 'inject-polyfill',
    async resolveId(source, importer, options) {
      if (source === POLYFILL_ID) {

        return { id: POLYFILL_ID, moduleSideEffects: true };
      }
      if (options.isEntry) {

        const resolution = await this.resolve(source, importer, { skipSelf: true, ...options });

        if (!resolution || resolution.external) return resolution;

        const moduleInfo = await this.load(resolution);

        moduleInfo.moduleSideEffects = true;

        return `${resolution.id}${PROXY_SUFFIX}`;
      }
      return null;
    },
    load(id) {
      if (id === POLYFILL_ID) {

        return "console.log('polyfill');";
      }
      if (id.endsWith(PROXY_SUFFIX)) {
        const entryId = id.slice(0, -PROXY_SUFFIX.length);

        const { hasDefaultExport } = this.getModuleInfo(entryId);
        let code =
          `import ${JSON.stringify(POLYFILL_ID)};` + `export * from ${JSON.stringify(entryId)};`;

        if (hasDefaultExport) {
          code += `export { default } from ${JSON.stringify(entryId)};`;
        }
        return code;
      }
      return null;
    }
  };
}
```

字段 值 (id) => string null Kind async, first Previous Hook 解析加载id的

。此外，这个钩子可以在任何时候从插件钩子中通过调用

来触发预加载与id对应的模块 Next Hook

可在未使用缓存或没有使用相同代码的缓存副本时转换加载的文件，否则应使用

- 定义自定义加载程序
- 返回 null会推迟到其他加载函数（最终是从文件系统加载的默认行为）
- 为了防止额外的解析开销，例如这个钩子已经使用了这个。parse出于某种原因，为了生成AST，这个钩子可以选择性地返回 {code，AST，map}对象。 ast必须是标准的 ESTree ast，每个节点都有开始和结束属性。如果转换不移动代码，可以通过将map设置为null来保留现有的sourcemaps。否则，您可能需要生成源映射。请参阅关于源代码转换的部分

字段 值 Type (code, id) => string Kind async, sequential Previous Hook

当前处理的文件的位置。如果使用了缓存，并且有该模块的缓存副本，那么如果插件为该钩子返回true，则应

Next Hook

一旦文件被处理和解析，模块就会被解析

- 可用于转换单个模块
- 为了防止额外的解析开销，例如这个钩子已经使用了 this.parse出于某种原因，为了生成AST
- 这个钩子可以选择性地返回 {code，AST，map}对象
- ast必须是标准的ESTree ast，每个节点都有 start和 end属性
- 如果转换不移动代码，可以通过将map设置为null来保留现有的sourcemaps。否则，您可能需要生成源映射。请参阅关于源代码转换的部分

```
npm install rollup-pluginutils @rollup/plugin-babel @babel/core @babel/preset-env  -D
```

```
plugins\rollup-plugin-babel.js
```js
import { createFilter } from 'rollup-pluginutils'
import babel from '@babel/core'
function plugin(pluginOptions = {}) {
  const defaultExtensions = ['.js', '.jsx']
  const { exclude, include, extensions = defaultExtensions } = pluginOptions;
  const extensionRegExp = new RegExp(`(${extensions.join('|')})
        )
  const userDefinedFilter = createFilter(include, exclude);
  const filter = id => extensionRegExp.test(id) && userDefinedFilter(id);
  return {
    name: 'babel',
    async transform(code, filename) {
      if (!filter(filename)) return null;
      let result = await babel.transformAsync(code);
      return result
    }
  }
}
export default plugin
```

字段 值 Type ({id, code, ast, resoledSources, moduleSideEffects, syntheticNamedExports) => boolean Kind: async, first Previous Hook

加载缓存文件以将其代码与缓存版本进行比较的位置 Next Hook

if no plugin returns true, otherwise

- 如果使用了 Rollup缓存（例如，在监视模式下或通过JavaScript API显式使用），如果在加载钩子之后，加载的代码与缓存副本的代码相同，则Rollup将跳过模块的转换钩子
- 为了防止这种情况，丢弃缓存的副本，而是转换一个模块，插件可以实现这个钩子并返回true。
- 这个钩子还可以用来找出缓存了哪些模块，并访问它们缓存的元信息
- 如果一个插件没有返回true，Rollup将触发其他插件的这个钩子，否则将跳过所有剩余的插件。

```
npx rollup -c -w
shouldTransformCachedModule
transform
moduleParsed

shouldTransformCachedModule
moduleParsed
```

字段 值 Type (moduleInfo: ModuleInfo) => void Kind: async, parallel Previous Hook

转换当前处理的文件的位置 Next Hook

并行解析所有发现的静态和动态导入（如果存在），否则buildEnd

- 每当模块被 Rollup完全解析时，就会调用这个钩子。看看 this.getModuleInfo了解传递给这个钩子的信息
- 与 transform钩子不同，这个钩子从不缓存，可以用来获取缓存模块和其他模块的信息，包括元属性的最终形状、代码和ast

字段 值 Type (specifier, importer) => string Kind async, first Previous Hook

已为导入文件分配模块 Next Hook

如果钩子使用尚未加载的id ,如果动态导入包含字符串且钩子未解析，请加载

，否则为

- 为动态导入定义自定义解析程序
- 返回 false表明导入应该保持原样，而不是传递给其他解析程序，从而使其成为外部的
- 与 resolveId钩子类似，还可以返回一个对象，将导入解析为不同的id，同时将其标记为外部
- 如果动态导入被传递一个字符串作为参数，那么从这个钩子返回的字符串将被解释为一个现有的模块id，而返回null将推迟到其它解析器resolveId

index.js

```
import('./msg.js').then(res => console.log(res))
```

字段 值 Type (error) => void Kind async, parallel Previous Hook moduleParsed, resolveId or resolveDynamicImport. Next Hook outputOptions 输出生成阶段的输出，因为这是构建阶段的最后一个挂钩

- 在 rollup完成打包时调用，但在调用 generate或 write之前调用；你也可以返回一个 Promise
- 如果在构建过程中发生错误，则会将其传递给此钩子
- 输出生成钩子可以提供有关生成的包的信息，并在完成后修改构建
- 输出生成阶段的第一个钩子是 outputOptions，最后一个钩子要么 generateBundle是通过成功生成输出
- 或者在输出生成过程中的任何时候发生错误 renderError
- 此外， closeBundle可以作为最后一个钩子调用，但用户有责任手动调用 bundle.close()以触发此钩子

plugins\rollup-plugin-generation.js
```

```
function generation() {
  return {
    name: 'rollup-plugin-generation',

    outputOptions(outputOptions) {
      console.log('outputOptions');
    },
    renderStart() {
      console.log('renderStart');
    },
    banner() {
      console.log('banner');
    },
    footer() {
      console.log('footer');
    },
    intro() {
      console.log('intro');
    },
    outro() {
      console.log('outro');
    },
    renderDynamicImport() {
      console.log('renderDynamicImport');
    },
    augmentChunkHash() {
      console.log('augmentChunkHash');
    },
    resolveFileUrl() {
      console.log('resolveFileUrl');
    },
    resolveImportMeta() {
      console.log('resolveImportMeta');
    },
    renderChunk() {
      console.log('renderChunk');
    },
    generateBundle() {
      console.log('generateBundle');
    },
    writeBundle() {
      console.log('writeBundle');
    },
    renderError() {
      console.log('renderError');
    },
    closeBundle() {
      console.log('closeBundle');
    }
  }
}
export default generation;
```

rollup.config.js

```
import build from './plugins/rollup-plugin-build.js';
+import generation from './plugins/rollup-plugin-generation.js';
export default {
  input: "./src/index.js",
  output: [{
    dir: 'dist',
  }],
  plugins: [
    build(),
+   generation()
  ]
}
```

字段 值 Type (outputOptions) => null Kind async, parallel Previous Hook

如果这是第一次生成输出，否则为

取决于先前生成的输出。这是输出生成阶段的第一个钩子 Next Hook

输出生成阶段的输出，因为这是构建阶段的最后一个挂钩

- 替换或操作传递给 `bundle.generate()` 的输出选项对象 `bundle.write()`
- 返回 `null` 并不能代替任何东西
- 如果您只需要读取输出选项，建议使用 `renderStart` 钩子，因为在考虑 `renderStart` 所有钩子的转换后，此钩子可以访问输出选项

字段 值 Type (outputOptions, inputOptions) => void 种类 async, parallel 上一个钩子 outputOptions 下一个钩子 banner, footer, intro and outro 并行运行

- 每次初始调用 `bundle.generate()` 或被 `bundle.write()` 调用
- 要在生成完成时收到通知，请使用 `generateBundle` 和 `renderError` 挂钩
- 当您需要访问传递给的输出选项时，建议使用此挂钩. `bundle.generate()` 或者 `bundle.write()` 因为它考虑了所有 `outputOptions` 挂钩的转换，并且还包含未设置选项的正确默认值。它还接收传递给的输入选项

字段 值 Type string (() => string) Kind async, parallel Previous Hook renderStart Next Hook 针对每个动态导入表达式

字段 值 Type string (() => string) Kind async, parallel Previous Hook renderStart Next Hook 针对每个动态导入表达式

字段 值 Type string (() => string) Kind async, parallel Previous Hook renderStart Next Hook 针对每个动态导入表达式

字段 值 Type string (() => string) Kind async, parallel Previous Hook renderStart Next Hook 针对每个动态导入表达式

字段 值 Type ({format, moduleId, targetModuleId, customResolution}) => {left: string, right: string} Kind async, parallel Previous Hook banner , footer, intro, outro Next Hook

对于每个在文件名中包含哈希的块

- 这个钩子提供了对如何呈现动态导入的细粒度控制
- 方法是替换导入表达式参数的左侧 (import() 和右侧 ( ) 的代码。)
- 返回null延迟到此类型的其他钩子并最终呈现特定于格式的默认值
- format是渲染的输出格式
- moduleId执行动态导入的模块的 id
- 如果导入可以解析为内部或外部 id，targetModuleId则将设置为此 id，否则将为null

plugins\rollup-plugin-renderDynamicImport.js

```
export default function dynamicImportPolyfillPlugin() {
  return {
    name: 'dynamic-import-polyfill',
    renderDynamicImport() {
      return {
        left: 'dynamicImportPolyfill(',
        right: ', import.meta.url)'
      };
    }
  };
}
```

```
dynamicImportPolyfill('./msg-ca034dda.js', import.meta.url).then(res => console.log(res.default));
function dynamicImportPolyfill(filename, url) {
  return new Promise((resolve) => {
    const script = document.createElement("script");
    script.type = "module";
    script.onload = () => {
      resolve(window.mod);
    };
    const absURL = new URL(filename, url).href;
    console.log(absURL);
    const blob = new Blob([
      `import * as mod from "${absURL}";`,
      ` window.mod = mod;`], { type: "text/javascript" });
    script.src = URL.createObjectURL(blob);
    document.head.appendChild(script);
  });
}
```

字段 值 Type (chunkInfo: ChunkInfo) => string Kind sync, sequential Previous Hook renderDynamicImport针对每个动态导入表达式 Next Hook

对于每次使用

所有其他访问

- 可用于增加单个块的散列
- 为每个 Rollup输出块调用
- 返回一个 false值不会修改散列
- 真实值将传递给 hash.update.
- 这 chunkInfo是 generateBundle不依赖文件名的属性的简化版本

字段 值 Type ({chunkId, fileName, format, moduleId, referenceId, relativePath}) => string Kind sync, first Previous Hook

对于在文件名中包含哈希的每个块 Next Hook

对于每个块

- 允许自定义Rollup如何解析插件通过此链接发出的文件的URL this.emitFile
- 默认情况下，Rollup 将生成代码
- import.meta.ROLLUP_FILE_URL_referenceId该代码应正确生成发出文件的绝对URL，而与输出格式和部署代码的主机系统无关

src\index.js

```
import logger from 'logger'
console.log(logger);
```

plugins\rollup-plugin-resolveFileUrl.js

```
export default function resolveFileUrl() {
  return {
    name: 'resolveFileUrl',
    resolveId(source) {
      if (source === 'logger') {
        return source;
      }
    },
    load(importee) {
      if (importee === 'logger') {
        let referenceId = this.emitFile({ type: 'asset', source: 'console.log("logger")', fileName: "logger.js" });
        return `export default import.meta.ROLLUP_FILE_URL_${referenceId}`;
      }
    },
    resolveFileUrl({ chunkId, fileName, format, moduleId, referenceId, relativePath }) {
      return `new URL('${fileName}', document.baseURI).href`;
    }
  };
}
```

字段 值 Type (property, {chunkId, moduleId, format}) => string Kind sync, first Previous Hook

对于在文件名中包含哈希的每个块 Next Hook

对于每个块

- 允许自定义 Rollup 如何处理 import.meta, import.meta.someProperty特别是 import.meta.url
- 在 ES 模块中， import.meta是一个对象， import.meta.url包含当前模块的 URL

对于 . 的每次使用

所有其他访问

Next Hook

- 可用于转换单个块
- 为每个 rollup输出块文件调用。返回null将不应用任何转换

字段 值 Type (options, bundle, isWrite) => void Kind async, sequential Previous Hook renderChunk对于每个块 Next Hook

如果输出是通过生成的，否则这是输出生成阶段的最后一个钩子，如果生成另一个输出

，可能会再次跟随

- 在 bundle.generate()之后调用
- 或者在 bundle.write()把文件写入之前调用
- 要在写入文件后修改文件，请使用 writeBundle挂钩
- writeBundle提供正在写入或生成的文件的完整列表及其详细信息

- 您可以通过从该钩子中的捆绑对象中删除文件来防止发出文件。要发出其他文件，请使用 `this.emitFile`插件上下文功能

```
npm i dedent
```

plugins\rollup-plugin-html.js

```
import dedent from 'dedent';
export default function html() {
  return {
    name: 'html',
    generateBundle(options, bundle) {
      let entryName;
      for (let fileName in bundle) {
        let assetOrChunkInfo = bundle[fileName];

        if (assetOrChunkInfo.isEntry) {
          entryName = fileName;
        }
      }
      this.emitFile({
        type: 'asset',
        fileName: 'index.html',
        source: dedent`

          rollup

          ${entryName}</span>" type="module">

        `
      });
    }
  };
}
```

字段 值 Type (options,bundle) => void Kind async, parallel Previous Hook generateBundle Next Hook 如果被调用，这是输出生成阶段的最后一个钩子，如果生成另一个输出，可能会再次跟随

- `bundle.write()`仅在写入所有文件后才调用
- 与 `generateBundle`钩子类似， `bundle`提供正在写入的文件的完整列表及其详细信息

字段 值 Type (error: Error) => void Kind async, parallel Previous Hook renderStart从到 的任何钩子renderChunk Next Hook

如果它被调用，这是输出生成阶段的最后一个钩子，如果生成另一个输出，可能会再次跟随

- `bundle.generate()`当 rollup 在或期间遇到错误时调用 `bundle.write()`
- 错误被传递给这个钩子。要在生成成功完成时收到通知，请使用 `generateBundle`钩子

字段 值 Type closeBundle: () => Promise void Kind async, parallel Previous Hook

如果有构建错误.否则何时bundle.close()被调用，在这种情况下，这将是最后一个被触发的钩子。

- 可用于清理可能正在运行的任何外部服务
- `Rollup` 的 `CLI` 将确保在每次运行后调用此钩子
- 但 `JavaScript API` 的用户有责任在 `bundle.close()`他们完成生成包后手动调用

3.Plugin Context

- thisemitfile (https://rollupjs.org/guide/en/#thisemitfile)
- Type: (emittedFile: EmittedChunk | EmittedAsset) => string
- 发出一个包含在生成输出中的新文件，并返回一个referenceId，该ID可在不同位置用于引用发出的文件
- emittedFile 可以有两种形式之一

```
type EmittedChunk = {
  type: 'chunk';
  id: string;
  name?: string;
  fileName?: string;
};

type EmittedAsset = {
  type: 'asset';
  name?: string;
  fileName?: string;
  source?: string | Uint8Array;
};
```

- thisload (https://rollupjs.org/guide/en/#thisload)
- Type: ({id: string, moduleSideEffects?: boolean | 'no-treeshake' | null, syntheticNamedExports?: boolean | string | null, meta?: {[plugin: string]: any} | null, resolveDependencies?: boolean}) => Promise
- 加载并解析与给定id对应的模块，并将附加的元信息附加到模块（如果提供）
- 这将触发与另一个模块导入该模块时相同的加载、转换和模块授权挂钩
- thisresolve (https://rollupjs.org/guide/en/#thisresolve)
- 使用Rollup使用的相同插件将导入解析为模块ID（即文件名），并确定导入是否应该是外部的
- 如果返回null，则无法通过Rollup或任何插件解析导入，但用户未明确将其标记为外部

4.实战案例

```
npm install  @rollup/plugin-commonjs   --save
```

src\index.js

```
import catValue from './cat.js';
console.log(catValue);
```

src\cat.js

```
module.exports = 'catValue';
```

plugins\rollup-plugin-commonjs.js

```
import { createFilter } from 'rollup-pluginutils'
import MagicString from 'magic-string';
import { walk } from 'estree-walker';
import path from 'path';
```

```
export default function (pluginOptions = {}) {
  const defaultExtensions = ['.js', '.jsx']
  const { exclude, include, extensions = defaultExtensions } = pluginOptions;
  const extensionRegExp = new RegExp(
    `(${extensions.join('|')})
  )
  const userDefinedFilter = createFilter(include, exclude);
  const filter = id => extensionRegExp.test(id) && userDefinedFilter(id);
  return {
    name: 'commonjs',
    transform(code, id) {
      if (!filter(id)) return null;
      const result = transformAndCheckExports(this.parse, code, id)
      return result;
    }
  }
}
function transformAndCheckExports(parse, code, id) {
  const { isEsModule, ast } = analyzeTopLevelStatements(parse, code, id);
  if (isEsModule) {
    return null;
  }
  return transformCommonjs(code, id, ast)
}
function getKeypath(node) {
  const parts = [];
  while (node.type === 'MemberExpression') {
    parts.unshift(node.property.name);
    node = node.object;
  }
  if (node.type !== 'Identifier') return null;
  const { name } = node;
  parts.unshift(name);
  return { name, keypath: parts.join('.') };
}
function analyzeTopLevelStatements(parse, code) {
  const ast = parse(code);
  let isEsModule = false;
  for (const node of ast.body) {
    switch (node.type) {
      case 'ExportDefaultDeclaration':
        isEsModule = true;
        break;
      case 'ExportNamedDeclaration':
        isEsModule = true;
        break;
      case 'ImportDeclaration':
        isEsModule = true;
        break;
      default:
    }
  }
  return { isEsModule, ast };
}
function transformCommonjs(code, id, ast) {
  const magicString = new MagicString(code);
  const exportDeclarations = [];
  let moduleExportsAssignment;
  walk(ast, {
    enter(node) {
      switch (node.type) {
        case 'AssignmentExpression':
          if (node.left.type === 'MemberExpression') {
            const flattened = getKeypath(node.left);
            if (flattened.keypath === 'module.exports') {
              moduleExportsAssignment = node;
            }
          }
          break;
        default:
          break;
      }
    }
  });
  const { left } = moduleExportsAssignment;
  const exportsName = path.basename(id, path.extname(id));
  magicString.overwrite(left.start, left.end, exportsName);
  magicString.prependRight(left.start, 'var ');
  exportDeclarations.push(`export default ${exportsName};`);
  const exportBlock = `\n\n${exportDeclarations.join('\n')}`;
  magicString.trim().append(exportBlock);
  return {
```

```
    code: magicString.toString()
  }
}
```

rollup.config.js

```
//import babel from '@rollup/plugin-babel'
//import babel from './plugins/rollup-plugin-babel.js'
//import commonjs from '@rollup/plugin-commonjs'
+import commonjs from './plugins/rollup-plugin-commonjs'
export default {
  input: "./src/index.js",
  output: {
    dir: 'dist'
  },
  plugins: [
    //babel(),
+   commonjs()
  ]
}
```

```
npm install @rollup/plugin-node-resolve check-is-array -D
```

```
(!) Unresolved dependencies
https:
isarray (imported by src/index.js)
```

src\index.js

```
import isArray from 'check-is-array';
console.log(isArray);
```

plugins\rollup-plugin-node-resolve.js

```
import path from 'path';
import Module from 'module';
function resolve() {
  return {
    name: 'resolve',

    async resolveId(importee, importer) {

      if (importee[0] === '.' || path.isAbsolute(importee)) {
        return null;
      }
      let location = Module.createRequire(path.dirname(importer)).resolve(importee);
      console.log(location);
      return location;
    }
  }
}
export default resolve;
```

rollup.config.js

```

import resolve from './plugins/rollup-plugin-node-resolve.js';

import alias from './plugins/rollup-plugin-alias.js';

export default {
  input: './src/index.js',

  output: {

    dir: 'dist'
  },
  plugins: [
    resolve(),
    alias({
      entries: [
        { find: './xx.js', replacement: 'check-is-array' }
      ]
    }),
  ],
  watch: {
    clearScreen: false
  }
}
```

plugins\rollup-plugin-alias.js

```
function matches(pattern, importee) {
    if (pattern instanceof RegExp) {
        return pattern.test(importee);
    }
    if (importee.length < pattern.length) {
        return false;
    }
    if (importee === pattern) {
        return true;
    }
    return importee.startsWith(pattern + '/');
}

function alias(options = {}) {
    const { entries } = options;
    if (entries.length === 0) {
        return {
            name: 'alias',
            resolveId: () => null
        };
    }
    return {
        name: 'alias',
        resolveId(importee, importer) {
            if (!importer) {
                return null;
            }
            const matchedEntry = entries.find((entry) => matches(entry.find, importee));
            if (!matchedEntry) {
                return null;
            }
            const updatedId = importee.replace(matchedEntry.find, matchedEntry.replacement);

            return this.resolve(updatedId, importer, Object.assign({ skipSelf: true }))
                .then((resolved) => resolved || { id: updatedId });
        }
    };
}
export default alias;
```