```
import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
        '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
      )
    }
  },
  renderChunk: {
    order,
    async handler(code, { fileName }, { file, dir }) {
      if (!filter(fileName)) return

      const { body } = parseAst(code, { lang: undefined }, fileName)
      const s = new MagicStringAST(code)
      let usingRequire = false

      for (const stmt of body) {
        if (stmt.type === 'ImportDeclaration') {
          if (stmt.importKind === 'type') continue

          const source = stmt.source.value

          const isBuiltinModule = builtinNodeModules && isBuiltin(source)

          const distFilename =
            file || (dir ? path.join(dir, fileName) : fileName)
          const importer = cwd
            ? path.resolve(cwd, distFilename)
            : distFilename
          const shouldProcess =
            isBuiltinModule ||
            ((await shouldTransform?.(source, importer)) ??
              (await isPureCJS(source, importer)))

          if (!shouldProcess) continue

          if (stmt.specifiers.length === 0) {
            // import 'cjs-module'
            if (isBuiltinModule) {
              // side-effect free
              s.removeNode(stmt)
            } else {
              // require('cjs-module')
              s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
              usingRequire = true
            }
            continue
```

```
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
    ])
  } else {
    // default import
    defaultId = specifier.local.name
  }
}

let requireCode: string
if (isBuiltinModule) {
  requireCode =
    source === 'process' || source === 'node:process'
      ? 'globalThis.process'
      : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
  requireCode = `__cjs_require(${JSON.stringify(source)})`
  usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
  defaultId ||= `_cjs_${namespaceId}_default`
}
if (defaultId) {
  // const name = require('cjs-module')
  codes.push(`const ${defaultId} = ${requireCode};`)
}
if (namespaceId) {
  // const ns = { ...default, default }
```

```
                            codes.push(
                                `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                            )
                        }
                        if (mapping.length > 0) {
                            codes.push(
                                `const {\n${mapping
                                    .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                                    .join(',\n')}\n} = ${defaultId || requireCode};`,
                            )
                        }
                        s.overwriteNode(stmt, codes.join('\n'))
                    }
                }

                if (usingRequire) {
                    const preamble = builtinNodeModules
                        ? `const ${REQUIRE} =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                        : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

                    if (code[0] === '#') {
                        // skip shebang line
                        const firstNewLineIndex = code.indexOf('\n') + 1
                        s.appendLeft(firstNewLineIndex, preamble)
                    } else {
                        s.prepend(preamble)
                    }
                }

                return generateTransform(s, fileName)
            },
        },
    }
}

export async function isPureCJS(
    id: string,
    importer: string,
): Promise<boolean> {
    if (!initted) {
        await init()
    }
```

```
  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                // import * as name from 'cjs-module'
                namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                if (specifier.importKind === 'type') continue
                // named import
                mapping.push([
                  s.sliceNode(specifier.imported),
                  specifier.local.name,
                ])
              } else {
                // default import
                defaultId = specifier.local.name
              }
            }

            let requireCode: string
            if (isBuiltinModule) {
              requireCode =
                source === 'process' || source === 'node:process'
                  ? 'globalThis.process'
                  : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
            } else {
              requireCode = `__cjs_require(${JSON.stringify(source)})`
              usingRequire = true
            }

            const codes: string[] = []
            if (namespaceId) {
              defaultId ||= `_cjs_${namespaceId}_default`
            }
            if (defaultId) {
              // const name = require('cjs-module')
              codes.push(`const ${defaultId} = ${requireCode};`)
            }
            if (namespaceId) {
              // const ns = { ...default, default }
              codes.push(
                `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
              )
            }
```

```
          if (mapping.length > 0) {
            codes.push(
              `const {\n${mapping
                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                .join(',\n')}\n} = ${defaultId || requireCode};`,
            )
          }
          s.overwriteNode(stmt, codes.join('\n'))
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

        if (code[0] === '#') {
          // skip shebang line
          const firstNewLineIndex = code.indexOf('\n') + 1
          s.appendLeft(firstNewLineIndex, preamble)
        } else {
          s.prepend(preamble)
        }
      }

      return generateTransform(s, fileName)
    },
  },
 }
}

export async function isPureCJS(
 id: string,
 importer: string,
): Promise<boolean> {
 if (!initted) {
  await init()
 }

 // ignore Node.js built-in modules, as their performance is comparable
 if (id.startsWith('node:')) return false
```

```
  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return
```

```
const { body } = parseAst(code, { lang: undefined }, fileName)
const s = new MagicStringAST(code)
let usingRequire = false

for (const stmt of body) {
  if (stmt.type === 'ImportDeclaration') {
    if (stmt.importKind === 'type') continue

    const source = stmt.source.value

    const isBuiltinModule = builtinNodeModules && isBuiltin(source)

    const distFilename =
      file || (dir ? path.join(dir, fileName) : fileName)
    const importer = cwd
      ? path.resolve(cwd, distFilename)
      : distFilename
    const shouldProcess =
      isBuiltinModule ||
      ((await shouldTransform?.(source, importer)) ??
        (await isPureCJS(source, importer)))

    if (!shouldProcess) continue

    if (stmt.specifiers.length === 0) {
      // import 'cjs-module'
      if (isBuiltinModule) {
        // side-effect free
        s.removeNode(stmt)
      } else {
        // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
      }
      continue
    }

    const mapping: [string, string][] = []
    let namespaceId: string | undefined
    let defaultId: string | undefined
    for (const specifier of stmt.specifiers) {
      // namespace
      if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
    } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
            s.sliceNode(specifier.imported),
            specifier.local.name,
        ])
    } else {
        // default import
        defaultId = specifier.local.name
    }
}

let requireCode: string
if (isBuiltinModule) {
    requireCode =
        source === 'process' || source === 'node:process'
            ? 'globalThis.process'
            : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
    requireCode = `__cjs_require(${JSON.stringify(source)})`
    usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
    defaultId ||= `_cjs_${namespaceId}_default`
}
if (defaultId) {
    // const name = require('cjs-module')
    codes.push(`const ${defaultId} = ${requireCode};`)
}
if (namespaceId) {
    // const ns = { ...default, default }
    codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
    )
}
if (mapping.length > 0) {
    codes.push(
        `const {\n${mapping
            .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
```

```
                  .join(',\n')}\n} = ${defaultId || requireCode};`,
              )
            }
            s.overwriteNode(stmt, codes.join('\n'))
          }
        }

        if (usingRequire) {
          const preamble = builtinNodeModules
            ? `const ${REQUIRE} =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })
```

```
        // different resolution, respect to original behavior
        if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
            return false
        }

        if (importResolved.endsWith('.cjs')) {
            return true
        } else if (importResolved.endsWith('.js')) {
            const pkgJsonPath = up({ cwd: importResolved })
            if (pkgJsonPath) {
                const pkgType = await getPackageType(pkgJsonPath)
                if (pkgType === 'module') return false
                if (pkgType === 'commonjs') return true
            }

            // detect by parsing
            const contents = await readFile(importResolved, 'utf8')
            try {
                parse(contents, importResolved)
                return true
            } catch {}
        }
    } catch {}
    return false
}

async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
    try {
        const pkg = JSON.parse(contents)
        return pkg.type as string | undefined
    } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'
```

```
export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false
```

```
for (const stmt of body) {
  if (stmt.type === 'ImportDeclaration') {
    if (stmt.importKind === 'type') continue

    const source = stmt.source.value

    const isBuiltinModule = builtinNodeModules && isBuiltin(source)

    const distFilename =
      file || (dir ? path.join(dir, fileName) : fileName)
    const importer = cwd
      ? path.resolve(cwd, distFilename)
      : distFilename
    const shouldProcess =
      isBuiltinModule ||
      ((await shouldTransform?.(source, importer)) ??
        (await isPureCJS(source, importer)))

    if (!shouldProcess) continue

    if (stmt.specifiers.length === 0) {
      // import 'cjs-module'
      if (isBuiltinModule) {
        // side-effect free
        s.removeNode(stmt)
      } else {
        // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
      }
      continue
    }

    const mapping: [string, string][] = []
    let namespaceId: string | undefined
    let defaultId: string | undefined
    for (const specifier of stmt.specifiers) {
      // namespace
      if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
      } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
          s.sliceNode(specifier.imported),
          specifier.local.name,
        ])
      } else {
        // default import
        defaultId = specifier.local.name
      }
    }

    let requireCode: string
    if (isBuiltinModule) {
      requireCode =
        source === 'process' || source === 'node:process'
          ? 'globalThis.process'
          : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
    } else {
      requireCode = `__cjs_require(${JSON.stringify(source)})`
      usingRequire = true
    }

    const codes: string[] = []
    if (namespaceId) {
      defaultId ||= `_cjs_${namespaceId}_default`
    }
    if (defaultId) {
      // const name = require('cjs-module')
      codes.push(`const ${defaultId} = ${requireCode};`)
    }
    if (namespaceId) {
      // const ns = { ...default, default }
      codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
      )
    }
    if (mapping.length > 0) {
      codes.push(
        `const {\n${mapping
          .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
          .join(',\n')}\n} = ${defaultId || requireCode};`,
      )
    }
    s.overwriteNode(stmt, codes.join('\n'))
```

```
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

        if (code[0] === '#') {
          // skip shebang line
          const firstNewLineIndex = code.indexOf('\n') + 1
          s.appendLeft(firstNewLineIndex, preamble)
        } else {
          s.prepend(preamble)
        }
      }

      return generateTransform(s, fileName)
    },
  },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }
    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false
```

```
const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue
```

```
const source = stmt.source.value

const isBuiltinModule = builtinNodeModules && isBuiltin(source)

const distFilename =
  file || (dir ? path.join(dir, fileName) : fileName)
const importer = cwd
  ? path.resolve(cwd, distFilename)
  : distFilename
const shouldProcess =
  isBuiltinModule ||
  ((await shouldTransform?.(source, importer)) ??
    (await isPureCJS(source, importer)))

if (!shouldProcess) continue

if (stmt.specifiers.length === 0) {
  // import 'cjs-module'
  if (isBuiltinModule) {
    // side-effect free
    s.removeNode(stmt)
  } else {
    // require('cjs-module')
    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
    usingRequire = true
  }
  continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
        ])
      } else {
        // default import
        defaultId = specifier.local.name
      }
    }

    let requireCode: string
    if (isBuiltinModule) {
      requireCode =
        source === 'process' || source === 'node:process'
          ? 'globalThis.process'
          : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
    } else {
      requireCode = `__cjs_require(${JSON.stringify(source)})`
      usingRequire = true
    }

    const codes: string[] = []
    if (namespaceId) {
      defaultId ||= `_cjs_${namespaceId}_default`
    }
    if (defaultId) {
      // const name = require('cjs-module')
      codes.push(`const ${defaultId} = ${requireCode};`)
    }
    if (namespaceId) {
      // const ns = { ...default, default }
      codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
      )
    }
    if (mapping.length > 0) {
      codes.push(
        `const {\n${mapping
          .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
          .join(',\n')}\n} = ${defaultId || requireCode};`,
      )
    }
    s.overwriteNode(stmt, codes.join('\n'))
  }
}

if (usingRequire) {
```

```
          const preamble = builtinNodeModules
            ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
            const pkgJsonPath = up({ cwd: importResolved })
            if (pkgJsonPath) {
                const pkgType = await getPackageType(pkgJsonPath)
                if (pkgType === 'module') return false
                if (pkgType === 'commonjs') return true
            }

            // detect by parsing
            const contents = await readFile(importResolved, 'utf8')
            try {
                parse(contents, importResolved)
                return true
            } catch {}
        }
    } catch {}
    return false
}

async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
    try {
        const pkg = JSON.parse(contents)
        return pkg.type as string | undefined
    } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)
            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                // import * as name from 'cjs-module'
                namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                if (specifier.importKind === 'type') continue
                // named import
                mapping.push([
                  s.sliceNode(specifier.imported),
                  specifier.local.name,
                ])
              } else {
                // default import
                defaultId = specifier.local.name
```

```
          }
        }

        let requireCode: string
        if (isBuiltinModule) {
          requireCode =
            source === 'process' || source === 'node:process'
              ? 'globalThis.process'
              : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
        } else {
          requireCode = `__cjs_require(${JSON.stringify(source)})`
          usingRequire = true
        }

        const codes: string[] = []
        if (namespaceId) {
          defaultId ||= `_cjs_${namespaceId}_default`
        }
        if (defaultId) {
          // const name = require('cjs-module')
          codes.push(`const ${defaultId} = ${requireCode};`)
        }
        if (namespaceId) {
          // const ns = { ...default, default }
          codes.push(
            `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
          )
        }
        if (mapping.length > 0) {
          codes.push(
            `const {\n${mapping
              .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
              .join(',\n')}\n} = ${defaultId || requireCode};`,
          )
        }
        s.overwriteNode(stmt, codes.join('\n'))
      }
    }

    if (usingRequire) {
      const preamble = builtinNodeModules
        ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
        : `import { createRequire as __cjs_createRequire } from "node:module";
```

```
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
```

```
          if (pkgType === 'commonjs') return true
        }

        // detect by parsing
        const contents = await readFile(importResolved, 'utf8')
        try {
          parse(contents, importResolved)
          return true
        } catch {}
      }
    } catch {}
    return false
  }

  async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
    try {
      const pkg = JSON.parse(contents)
      return pkg.type as string | undefined
    } catch {}
  }import { readFile } from 'node:fs/promises'
  import { isBuiltin } from 'node:module'
  import path from 'node:path'
  import process from 'node:process'
  import { init, parse } from 'cjs-module-lexer'
  import { up } from 'empathic/package'
  import { generateTransform, MagicStringAST } from 'magic-string-ast'
  import { resolvePathSync } from 'mlly'
  import { parseAst } from 'rolldown/parseAst'
  import { createFilter } from 'unplugin-utils'
  import { resolveOptions, type Options } from './options'
  import type { Plugin } from 'rolldown'

  export * from './options'

  let initted = false

  const REQUIRE = `__cjs_require`

  export function RequireCJS(userOptions: Options = {}): Plugin {
    const { include, exclude, order, shouldTransform, builtinNodeModules } =
      resolveOptions(userOptions)
    const filter = createFilter(include, exclude)
    let cwd: string
    return {
      name: 'rolldown-plugin-require-cjs',
      async buildStart() {
        if (!initted) {
          await init()
          initted = true
        }
      },
      options(options) {
        if (options.platform !== 'node') {
          this.error(
            '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment. Please make sure to set `platform: "node"` in the options.',
          )
        }
        cwd = options.cwd || process.cwd()
      },
      outputOptions(options) {
        if (!['es', 'esm', 'module'].includes(options.format as any)) {
          throw new Error(
            '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
          )
        }
      },
      renderChunk: {
        order,
        async handler(code, { fileName }, { file, dir }) {
          if (!filter(fileName)) return

          const { body } = parseAst(code, { lang: undefined }, fileName)
          const s = new MagicStringAST(code)
          let usingRequire = false

          for (const stmt of body) {
            if (stmt.type === 'ImportDeclaration') {
              if (stmt.importKind === 'type') continue

              const source = stmt.source.value

              const isBuiltinModule = builtinNodeModules && isBuiltin(source)

              const distFilename =
                file || (dir ? path.join(dir, fileName) : fileName)
              const importer = cwd
                ? path.resolve(cwd, distFilename)
                : distFilename
              const shouldProcess =
                isBuiltinModule ||
                ((await shouldTransform?.(source, importer)) ??
                    (await isPureCJS(source, importer)))

              if (!shouldProcess) continue

              if (stmt.specifiers.length === 0) {
                // import 'cjs-module'
                if (isBuiltinModule) {
                    // side-effect free
                    s.removeNode(stmt)
                } else {
                    // require('cjs-module')
                    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                    usingRequire = true
                }
                continue
              }

              const mapping: [string, string][] = []
              let namespaceId: string | undefined
              let defaultId: string | undefined
              for (const specifier of stmt.specifiers) {
                // namespace
                if (specifier.type === 'ImportNamespaceSpecifier') {
                    // import * as name from 'cjs-module'
                    namespaceId = specifier.local.name
                } else if (specifier.type === 'ImportSpecifier') {
                    if (specifier.importKind === 'type') continue
                    // named import
                    mapping.push([
                        s.sliceNode(specifier.imported),
                        specifier.local.name,
                    ])
                } else {
                    // default import
                    defaultId = specifier.local.name
                }
              }

              let requireCode: string
```

```
            if (isBuiltinModule) {
              requireCode =
                source === 'process' || source === 'node:process'
                  ? 'globalThis.process'
                  : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
            } else {
              requireCode = `__cjs_require(${JSON.stringify(source)})`
              usingRequire = true
            }

            const codes: string[] = []
            if (namespaceId) {
              defaultId ||= `_cjs_${namespaceId}_default`
            }
            if (defaultId) {
              // const name = require('cjs-module')
              codes.push(`const ${defaultId} = ${requireCode};`)
            }
            if (namespaceId) {
              // const ns = { ...default, default }
              codes.push(
                `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
              )
            }
            if (mapping.length > 0) {
              codes.push(
                `const {\n${mapping
                  .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                  .join(',\n')}\n} = ${defaultId || requireCode};`,
              )
            }
            s.overwriteNode(stmt, codes.join('\n'))
          }
        }

        if (usingRequire) {
          const preamble = builtinNodeModules
            ? `const ${REQUIRE} =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
```

```
                    const firstNewLineIndex = code.indexOf('\n') + 1
                    s.appendLeft(firstNewLineIndex, preamble)
                } else {
                    s.prepend(preamble)
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment. Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                // import * as name from 'cjs-module'
                namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                if (specifier.importKind === 'type') continue
                // named import
                mapping.push([
                  s.sliceNode(specifier.imported),
                  specifier.local.name,
                ])
              } else {
                // default import
                defaultId = specifier.local.name
              }
            }

            let requireCode: string
            if (isBuiltinModule) {
              requireCode =
                source === 'process' || source === 'node:process'
                  ? 'globalThis.process'
```

```
                  : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
            } else {
              requireCode = `__cjs_require(${JSON.stringify(source)})`
              usingRequire = true
            }

            const codes: string[] = []
            if (namespaceId) {
              defaultId ||= `_cjs_${namespaceId}_default`
            }
            if (defaultId) {
              // const name = require('cjs-module')
              codes.push(`const ${defaultId} = ${requireCode};`)
            }
            if (namespaceId) {
              // const ns = { ...default, default }
              codes.push(
                `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
              )
            }
            if (mapping.length > 0) {
              codes.push(
                `const {\n${mapping
                  .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                  .join(',\n')}\n} = ${defaultId || requireCode};`,
              )
            }
            s.overwriteNode(stmt, codes.join('\n'))
          }
        }

        if (usingRequire) {
          const preamble = builtinNodeModules
            ?                `const                ${REQUIRE}                =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
```

```
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment. Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue
```

```
if (stmt.specifiers.length === 0) {
  // import 'cjs-module'
  if (isBuiltinModule) {
    // side-effect free
    s.removeNode(stmt)
  } else {
    // require('cjs-module')
    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
    usingRequire = true
  }
  continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
    ])
  } else {
    // default import
    defaultId = specifier.local.name
  }
}

let requireCode: string
if (isBuiltinModule) {
  requireCode =
    source === 'process' || source === 'node:process'
      ? 'globalThis.process'
      : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
  requireCode = `__cjs_require(${JSON.stringify(source)})`
  usingRequire = true
```

```
          }

          const codes: string[] = []
          if (namespaceId) {
            defaultId ||= `_cjs_${namespaceId}_default`
          }
          if (defaultId) {
            // const name = require('cjs-module')
            codes.push(`const ${defaultId} = ${requireCode};`)
          }
          if (namespaceId) {
            // const ns = { ...default, default }
            codes.push(
              `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
            )
          }
          if (mapping.length > 0) {
            codes.push(
              `const {\n${mapping
                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                .join(',\n')}\n} = ${defaultId || requireCode};`,
            )
          }
          s.overwriteNode(stmt, codes.join('\n'))
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

        if (code[0] === '#') {
          // skip shebang line
          const firstNewLineIndex = code.indexOf('\n') + 1
          s.appendLeft(firstNewLineIndex, preamble)
        } else {
          s.prepend(preamble)
        }
      }

      return generateTransform(s, fileName)
```

```
          },
        },
      }
    }

    export async function isPureCJS(
      id: string,
      importer: string,
    ): Promise<boolean> {
      if (!initted) {
        await init()
      }

      // ignore Node.js built-in modules, as their performance is comparable
      if (id.startsWith('node:')) return false

      try {
        const importResolved = resolvePathSync(id, { url: importer })
        const requireResolved = require.resolve(id, { paths: [importer] })

        // different resolution, respect to original behavior
        if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
          return false
        }

        if (importResolved.endsWith('.cjs')) {
          return true
        } else if (importResolved.endsWith('.js')) {
          const pkgJsonPath = up({ cwd: importResolved })
          if (pkgJsonPath) {
            const pkgType = await getPackageType(pkgJsonPath)
            if (pkgType === 'module') return false
            if (pkgType === 'commonjs') return true
          }

          // detect by parsing
          const contents = await readFile(importResolved, 'utf8')
          try {
            parse(contents, importResolved)
            return true
          } catch {}
        }
      } catch {}
      return false
```

```
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
```

```
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
```

```
    // side-effect free
    s.removeNode(stmt)
  } else {
    // require('cjs-module')
    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
    usingRequire = true
  }
  continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
    ])
  } else {
    // default import
    defaultId = specifier.local.name
  }
}

let requireCode: string
if (isBuiltinModule) {
  requireCode =
    source === 'process' || source === 'node:process'
      ? 'globalThis.process'
      : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
  requireCode = `__cjs_require(${JSON.stringify(source)})`
  usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
```

```
                        defaultId ||= `_cjs_${namespaceId}_default`
                    }
                    if (defaultId) {
                        // const name = require('cjs-module')
                        codes.push(`const ${defaultId} = ${requireCode};`)
                    }
                    if (namespaceId) {
                        // const ns = { ...default, default }
                        codes.push(
                            `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                        )
                    }
                    if (mapping.length > 0) {
                        codes.push(
                            `const {\n${mapping
                                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                                .join(',\n')}\n} = ${defaultId || requireCode};`,
                        )
                    }
                    s.overwriteNode(stmt, codes.join('\n'))
                }
            }

            if (usingRequire) {
                const preamble = builtinNodeModules
                    ?                                   `const                                   ${REQUIRE}                                   =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                    : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

                if (code[0] === '#') {
                    // skip shebang line
                    const firstNewLineIndex = code.indexOf('\n') + 1
                    s.appendLeft(firstNewLineIndex, preamble)
                } else {
                    s.prepend(preamble)
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}
```

```
export async function isPureCJS(
    id: string,
    importer: string,
): Promise<boolean> {
    if (!initted) {
        await init()
    }

    // ignore Node.js built-in modules, as their performance is comparable
    if (id.startsWith('node:')) return false

    try {
        const importResolved = resolvePathSync(id, { url: importer })
        const requireResolved = require.resolve(id, { paths: [importer] })

        // different resolution, respect to original behavior
        if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
            return false
        }

        if (importResolved.endsWith('.cjs')) {
            return true
        } else if (importResolved.endsWith('.js')) {
            const pkgJsonPath = up({ cwd: importResolved })
            if (pkgJsonPath) {
                const pkgType = await getPackageType(pkgJsonPath)
                if (pkgType === 'module') return false
                if (pkgType === 'commonjs') return true
            }

            // detect by parsing
            const contents = await readFile(importResolved, 'utf8')
            try {
                parse(contents, importResolved)
                return true
            } catch {}
        }
    } catch {}
    return false
}

async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
```

```
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
```

```
},
outputOptions(options) {
  if (!['es', 'esm', 'module'].includes(options.format as any)) {
    throw new Error(
      '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
    )
  }
},
renderChunk: {
  order,
  async handler(code, { fileName }, { file, dir }) {
    if (!filter(fileName)) return

    const { body } = parseAst(code, { lang: undefined }, fileName)
    const s = new MagicStringAST(code)
    let usingRequire = false

    for (const stmt of body) {
      if (stmt.type === 'ImportDeclaration') {
        if (stmt.importKind === 'type') continue

        const source = stmt.source.value

        const isBuiltinModule = builtinNodeModules && isBuiltin(source)

        const distFilename =
          file || (dir ? path.join(dir, fileName) : fileName)
        const importer = cwd
          ? path.resolve(cwd, distFilename)
          : distFilename
        const shouldProcess =
          isBuiltinModule ||
          ((await shouldTransform?.(source, importer)) ??
            (await isPureCJS(source, importer)))

        if (!shouldProcess) continue

        if (stmt.specifiers.length === 0) {
          // import 'cjs-module'
          if (isBuiltinModule) {
            // side-effect free
            s.removeNode(stmt)
          } else {
            // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
    }
    continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
    // namespace
    if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
    } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
            s.sliceNode(specifier.imported),
            specifier.local.name,
        ])
    } else {
        // default import
        defaultId = specifier.local.name
    }
}

let requireCode: string
if (isBuiltinModule) {
    requireCode =
        source === 'process' || source === 'node:process'
            ? 'globalThis.process'
            : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
    requireCode = `__cjs_require(${JSON.stringify(source)})`
    usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
    defaultId ||= `_cjs_${namespaceId}_default`
}
if (defaultId) {
    // const name = require('cjs-module')
```

```
                    codes.push(`const ${defaultId} = ${requireCode};`)
                  }
                  if (namespaceId) {
                    // const ns = { ...default, default }
                    codes.push(
                      `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                    )
                  }
                  if (mapping.length > 0) {
                    codes.push(
                      `const {\n${mapping
                        .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                        .join(',\n')}\n} = ${defaultId || requireCode};`,
                    )
                  }
                  s.overwriteNode(stmt, codes.join('\n'))
                }
              }

              if (usingRequire) {
                const preamble = builtinNodeModules
                  ?                 `const                 ${REQUIRE}                 =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                  : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

                if (code[0] === '#') {
                  // skip shebang line
                  const firstNewLineIndex = code.indexOf('\n') + 1
                  s.appendLeft(firstNewLineIndex, preamble)
                } else {
                  s.prepend(preamble)
                }
              }

              return generateTransform(s, fileName)
            },
          },
        }
      }

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
```

```
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
```

```
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
    ])
  } else {
    // default import
    defaultId = specifier.local.name
  }
}

let requireCode: string
if (isBuiltinModule) {
  requireCode =
    source === 'process' || source === 'node:process'
      ? 'globalThis.process'
      : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
  requireCode = `__cjs_require(${JSON.stringify(source)})`
  usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
  defaultId ||= `_cjs_${namespaceId}_default`
}
if (defaultId) {
  // const name = require('cjs-module')
  codes.push(`const ${defaultId} = ${requireCode};`)
}
if (namespaceId) {
  // const ns = { ...default, default }
```

```
                    codes.push(
                        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                    )
                }
                if (mapping.length > 0) {
                    codes.push(
                        `const {\n${mapping
                            .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                            .join(',\n')}\n} = ${defaultId || requireCode};`,
                    )
                }
                s.overwriteNode(stmt, codes.join('\n'))
            }
        }

        if (usingRequire) {
            const preamble = builtinNodeModules
                ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

            if (code[0] === '#') {
                // skip shebang line
                const firstNewLineIndex = code.indexOf('\n') + 1
                s.appendLeft(firstNewLineIndex, preamble)
            } else {
                s.prepend(preamble)
            }
        }

        return generateTransform(s, fileName)
    },
  },
 }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }
  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                // import * as name from 'cjs-module'
                namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                if (specifier.importKind === 'type') continue
                // named import
                mapping.push([
                  s.sliceNode(specifier.imported),
                  specifier.local.name,
                ])
              } else {
                // default import
                defaultId = specifier.local.name
              }
            }

            let requireCode: string
            if (isBuiltinModule) {
              requireCode =
                source === 'process' || source === 'node:process'
                  ? 'globalThis.process'
                  : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
            } else {
              requireCode = `__cjs_require(${JSON.stringify(source)})`
              usingRequire = true
            }

            const codes: string[] = []
            if (namespaceId) {
              defaultId ||= `_cjs_${namespaceId}_default`
            }
            if (defaultId) {
              // const name = require('cjs-module')
              codes.push(`const ${defaultId} = ${requireCode};`)
            }
            if (namespaceId) {
              // const ns = { ...default, default }
              codes.push(
                `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
              )
            }
```

```
                    if (mapping.length > 0) {
                        codes.push(
                            `const {\n${mapping
                                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                                .join(',\n')}\n} = ${defaultId || requireCode};`,
                        )
                    }
                    s.overwriteNode(stmt, codes.join('\n'))
                }
            }

            if (usingRequire) {
                const preamble = builtinNodeModules
                    ?                             `const                              ${REQUIRE}                              =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                    : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

                if (code[0] === '#') {
                    // skip shebang line
                    const firstNewLineIndex = code.indexOf('\n') + 1
                    s.appendLeft(firstNewLineIndex, preamble)
                } else {
                    s.prepend(preamble)
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false
  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return
```

```
const { body } = parseAst(code, { lang: undefined }, fileName)
const s = new MagicStringAST(code)
let usingRequire = false

for (const stmt of body) {
  if (stmt.type === 'ImportDeclaration') {
    if (stmt.importKind === 'type') continue

    const source = stmt.source.value

    const isBuiltinModule = builtinNodeModules && isBuiltin(source)

    const distFilename =
      file || (dir ? path.join(dir, fileName) : fileName)
    const importer = cwd
      ? path.resolve(cwd, distFilename)
      : distFilename
    const shouldProcess =
      isBuiltinModule ||
      ((await shouldTransform?.(source, importer)) ??
        (await isPureCJS(source, importer)))

    if (!shouldProcess) continue

    if (stmt.specifiers.length === 0) {
      // import 'cjs-module'
      if (isBuiltinModule) {
        // side-effect free
        s.removeNode(stmt)
      } else {
        // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
      }
      continue
    }

    const mapping: [string, string][] = []
    let namespaceId: string | undefined
    let defaultId: string | undefined
    for (const specifier of stmt.specifiers) {
      // namespace
      if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
    } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
            s.sliceNode(specifier.imported),
            specifier.local.name,
        ])
    } else {
        // default import
        defaultId = specifier.local.name
    }
}

let requireCode: string
if (isBuiltinModule) {
    requireCode =
        source === 'process' || source === 'node:process'
            ? 'globalThis.process'
            : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
    requireCode = `__cjs_require(${JSON.stringify(source)})`
    usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
    defaultId ||= `_cjs_${namespaceId}_default`
}
if (defaultId) {
    // const name = require('cjs-module')
    codes.push(`const ${defaultId} = ${requireCode};`)
}
if (namespaceId) {
    // const ns = { ...default, default }
    codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
    )
}
if (mapping.length > 0) {
    codes.push(
        `const {\n${mapping
            .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
```

```
                        .join(',\n')}\n} = ${defaultId || requireCode};`,
                    )
                }
                s.overwriteNode(stmt, codes.join('\n'))
            }
        }

        if (usingRequire) {
            const preamble = builtinNodeModules
                ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

            if (code[0] === '#') {
                // skip shebang line
                const firstNewLineIndex = code.indexOf('\n') + 1
                s.appendLeft(firstNewLineIndex, preamble)
            } else {
                s.prepend(preamble)
            }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })
```

```
        // different resolution, respect to original behavior
        if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
          return false
        }

        if (importResolved.endsWith('.cjs')) {
          return true
        } else if (importResolved.endsWith('.js')) {
          const pkgJsonPath = up({ cwd: importResolved })
          if (pkgJsonPath) {
            const pkgType = await getPackageType(pkgJsonPath)
            if (pkgType === 'module') return false
            if (pkgType === 'commonjs') return true
          }

          // detect by parsing
          const contents = await readFile(importResolved, 'utf8')
          try {
            parse(contents, importResolved)
            return true
          } catch {}
        }
      } catch {}
      return false
    }

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'
```

```
export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false
```

```
for (const stmt of body) {
  if (stmt.type === 'ImportDeclaration') {
    if (stmt.importKind === 'type') continue

    const source = stmt.source.value

    const isBuiltinModule = builtinNodeModules && isBuiltin(source)

    const distFilename =
      file || (dir ? path.join(dir, fileName) : fileName)
    const importer = cwd
      ? path.resolve(cwd, distFilename)
      : distFilename
    const shouldProcess =
      isBuiltinModule ||
      ((await shouldTransform?.(source, importer)) ??
        (await isPureCJS(source, importer)))

    if (!shouldProcess) continue

    if (stmt.specifiers.length === 0) {
      // import 'cjs-module'
      if (isBuiltinModule) {
        // side-effect free
        s.removeNode(stmt)
      } else {
        // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
      }
      continue
    }

    const mapping: [string, string][] = []
    let namespaceId: string | undefined
    let defaultId: string | undefined
    for (const specifier of stmt.specifiers) {
      // namespace
      if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
      } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
          s.sliceNode(specifier.imported),
          specifier.local.name,
        ])
      } else {
        // default import
        defaultId = specifier.local.name
      }
    }

    let requireCode: string
    if (isBuiltinModule) {
      requireCode =
        source === 'process' || source === 'node:process'
          ? 'globalThis.process'
          : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
    } else {
      requireCode = `__cjs_require(${JSON.stringify(source)})`
      usingRequire = true
    }

    const codes: string[] = []
    if (namespaceId) {
      defaultId ||= `_cjs_${namespaceId}_default`
    }
    if (defaultId) {
      // const name = require('cjs-module')
      codes.push(`const ${defaultId} = ${requireCode};`)
    }
    if (namespaceId) {
      // const ns = { ...default, default }
      codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
      )
    }
    if (mapping.length > 0) {
      codes.push(
        `const {\n${mapping
          .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
          .join(',\n')}\n} = ${defaultId || requireCode};`,
      )
    }
    s.overwriteNode(stmt, codes.join('\n'))
```

```
          }
        }

        if (usingRequire) {
          const preamble = builtinNodeModules
            ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }
    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false
```

```
const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue
```

```
const source = stmt.source.value

const isBuiltinModule = builtinNodeModules && isBuiltin(source)

const distFilename =
  file || (dir ? path.join(dir, fileName) : fileName)
const importer = cwd
  ? path.resolve(cwd, distFilename)
  : distFilename
const shouldProcess =
  isBuiltinModule ||
  ((await shouldTransform?.(source, importer)) ??
    (await isPureCJS(source, importer)))

if (!shouldProcess) continue

if (stmt.specifiers.length === 0) {
  // import 'cjs-module'
  if (isBuiltinModule) {
    // side-effect free
    s.removeNode(stmt)
  } else {
    // require('cjs-module')
    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
    usingRequire = true
  }
  continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
        ])
      } else {
        // default import
        defaultId = specifier.local.name
      }
    }

    let requireCode: string
    if (isBuiltinModule) {
      requireCode =
        source === 'process' || source === 'node:process'
          ? 'globalThis.process'
          : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
    } else {
      requireCode = `__cjs_require(${JSON.stringify(source)})`
      usingRequire = true
    }

    const codes: string[] = []
    if (namespaceId) {
      defaultId ||= `_cjs_${namespaceId}_default`
    }
    if (defaultId) {
      // const name = require('cjs-module')
      codes.push(`const ${defaultId} = ${requireCode};`)
    }
    if (namespaceId) {
      // const ns = { ...default, default }
      codes.push(
        `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
      )
    }
    if (mapping.length > 0) {
      codes.push(
        `const {\n${mapping
          .map(([k, v]) => `  ${k === v ? v : `${k}: ${v}`}`)
          .join(',\n')}\n} = ${defaultId || requireCode};`,
      )
    }
    s.overwriteNode(stmt, codes.join('\n'))
  }
}

if (usingRequire) {
```

```
          const preamble = builtinNodeModules
            ? `const ${REQUIRE} =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
            : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
            const pkgJsonPath = up({ cwd: importResolved })
            if (pkgJsonPath) {
                const pkgType = await getPackageType(pkgJsonPath)
                if (pkgType === 'module') return false
                if (pkgType === 'commonjs') return true
            }

            // detect by parsing
            const contents = await readFile(importResolved, 'utf8')
            try {
                parse(contents, importResolved)
                return true
            } catch {}
        }
    } catch {}
    return false
}

async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
    try {
        const pkg = JSON.parse(contents)
        return pkg.type as string | undefined
    } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)
            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                // side-effect free
                s.removeNode(stmt)
              } else {
                // require('cjs-module')
                s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                // import * as name from 'cjs-module'
                namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                if (specifier.importKind === 'type') continue
                // named import
                mapping.push([
                  s.sliceNode(specifier.imported),
                  specifier.local.name,
                ])
              } else {
                // default import
                defaultId = specifier.local.name
```

```
            }
          }

          let requireCode: string
          if (isBuiltinModule) {
            requireCode =
              source === 'process' || source === 'node:process'
                ? 'globalThis.process'
                : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
          } else {
            requireCode = `__cjs_require(${JSON.stringify(source)})`
            usingRequire = true
          }

          const codes: string[] = []
          if (namespaceId) {
            defaultId ||= `_cjs_${namespaceId}_default`
          }
          if (defaultId) {
            // const name = require('cjs-module')
            codes.push(`const ${defaultId} = ${requireCode};`)
          }
          if (namespaceId) {
            // const ns = { ...default, default }
            codes.push(
              `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
            )
          }
          if (mapping.length > 0) {
            codes.push(
              `const {\n${mapping
                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                .join(',\n')}\n} = ${defaultId || requireCode};`,
            )
          }
          s.overwriteNode(stmt, codes.join('\n'))
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
```

```
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

          if (code[0] === '#') {
            // skip shebang line
            const firstNewLineIndex = code.indexOf('\n') + 1
            s.appendLeft(firstNewLineIndex, preamble)
          } else {
            s.prepend(preamble)
          }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
```

```
          if (pkgType === 'commonjs') return true
        }

        // detect by parsing
        const contents = await readFile(importResolved, 'utf8')
        try {
          parse(contents, importResolved)
          return true
        } catch {}
      }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
        if (!initted) {
          await init()
          initted = true
        }
    },
    options(options) {
        if (options.platform !== 'node') {
          this.error(
            '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
          )
        }
        cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
        if (!['es', 'esm', 'module'].includes(options.format as any)) {
          throw new Error(
            '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
          )
        }
    },
    renderChunk: {
        order,
        async handler(code, { fileName }, { file, dir }) {
          if (!filter(fileName)) return

          const { body } = parseAst(code, { lang: undefined }, fileName)
          const s = new MagicStringAST(code)
          let usingRequire = false

          for (const stmt of body) {
            if (stmt.type === 'ImportDeclaration') {
              if (stmt.importKind === 'type') continue

              const source = stmt.source.value

              const isBuiltinModule = builtinNodeModules && isBuiltin(source)

              const distFilename =
                file || (dir ? path.join(dir, fileName) : fileName)
              const importer = cwd
                ? path.resolve(cwd, distFilename)
                : distFilename
              const shouldProcess =
                isBuiltinModule ||
                ((await shouldTransform?.(source, importer)) ??
                    (await isPureCJS(source, importer)))

              if (!shouldProcess) continue

              if (stmt.specifiers.length === 0) {
                // import 'cjs-module'
                if (isBuiltinModule) {
                    // side-effect free
                    s.removeNode(stmt)
                } else {
                    // require('cjs-module')
                    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                    usingRequire = true
                }
                continue
              }

              const mapping: [string, string][] = []
              let namespaceId: string | undefined
              let defaultId: string | undefined
              for (const specifier of stmt.specifiers) {
                // namespace
                if (specifier.type === 'ImportNamespaceSpecifier') {
                    // import * as name from 'cjs-module'
                    namespaceId = specifier.local.name
                } else if (specifier.type === 'ImportSpecifier') {
                    if (specifier.importKind === 'type') continue
                    // named import
                    mapping.push([
                        s.sliceNode(specifier.imported),
                        specifier.local.name,
                    ])
                } else {
                    // default import
                    defaultId = specifier.local.name
                }
              }

              let requireCode: string
```

```
        if (isBuiltinModule) {
          requireCode =
            source === 'process' || source === 'node:process'
              ? 'globalThis.process'
              : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
        } else {
          requireCode = `__cjs_require(${JSON.stringify(source)})`
          usingRequire = true
        }

        const codes: string[] = []
        if (namespaceId) {
          defaultId ||= `_cjs_${namespaceId}_default`
        }
        if (defaultId) {
          // const name = require('cjs-module')
          codes.push(`const ${defaultId} = ${requireCode};`)
        }
        if (namespaceId) {
          // const ns = { ...default, default }
          codes.push(
            `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
          )
        }
        if (mapping.length > 0) {
          codes.push(
            `const {\n${mapping
              .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
              .join(',\n')}\n} = ${defaultId || requireCode};`,
          )
        }
        s.overwriteNode(stmt, codes.join('\n'))
      }
    }

    if (usingRequire) {
      const preamble = builtinNodeModules
        ? `const ${REQUIRE} =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
        : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

      if (code[0] === '#') {
        // skip shebang line
```

```
                const firstNewLineIndex = code.indexOf('\n') + 1
                s.appendLeft(firstNewLineIndex, preamble)
            } else {
                s.prepend(preamble)
            }
        }

        return generateTransform(s, fileName)
      },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment. Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                  (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
                  // side-effect free
                  s.removeNode(stmt)
              } else {
                  // require('cjs-module')
                  s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
                  usingRequire = true
              }
              continue
            }

            const mapping: [string, string][] = []
            let namespaceId: string | undefined
            let defaultId: string | undefined
            for (const specifier of stmt.specifiers) {
              // namespace
              if (specifier.type === 'ImportNamespaceSpecifier') {
                  // import * as name from 'cjs-module'
                  namespaceId = specifier.local.name
              } else if (specifier.type === 'ImportSpecifier') {
                  if (specifier.importKind === 'type') continue
                  // named import
                  mapping.push([
                      s.sliceNode(specifier.imported),
                      specifier.local.name,
                  ])
              } else {
                  // default import
                  defaultId = specifier.local.name
              }
            }

            let requireCode: string
            if (isBuiltinModule) {
              requireCode =
                  source === 'process' || source === 'node:process'
                      ? 'globalThis.process'
```

```
                : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
          } else {
            requireCode = `__cjs_require(${JSON.stringify(source)})`
            usingRequire = true
          }

          const codes: string[] = []
          if (namespaceId) {
            defaultId ||= `_cjs_${namespaceId}_default`
          }
          if (defaultId) {
            // const name = require('cjs-module')
            codes.push(`const ${defaultId} = ${requireCode};`)
          }
          if (namespaceId) {
            // const ns = { ...default, default }
            codes.push(
              `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
            )
          }
          if (mapping.length > 0) {
            codes.push(
              `const {\n${mapping
                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                .join(',\n')}\n} = ${defaultId || requireCode};`,
            )
          }
          s.overwriteNode(stmt, codes.join('\n'))
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

        if (code[0] === '#') {
          // skip shebang line
          const firstNewLineIndex = code.indexOf('\n') + 1
          s.appendLeft(firstNewLineIndex, preamble)
        } else {
          s.prepend(preamble)
```

```
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
        } catch {}
      }
    } catch {}
    return false
}

async function getPackageType(path: string): Promise<string | undefined> {
    const contents = await readFile(path, 'utf8')
    try {
        const pkg = JSON.parse(contents)
        return pkg.type as string | undefined
    } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
    const { include, exclude, order, shouldTransform, builtinNodeModules } =
        resolveOptions(userOptions)
    const filter = createFilter(include, exclude)
    let cwd: string
    return {
        name: 'rolldown-plugin-require-cjs',
        async buildStart() {
            if (!initted) {
                await init()
                initted = true
            }
        },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment. Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue
```

```
if (stmt.specifiers.length === 0) {
  // import 'cjs-module'
  if (isBuiltinModule) {
    // side-effect free
    s.removeNode(stmt)
  } else {
    // require('cjs-module')
    s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
    usingRequire = true
  }
  continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
  // namespace
  if (specifier.type === 'ImportNamespaceSpecifier') {
    // import * as name from 'cjs-module'
    namespaceId = specifier.local.name
  } else if (specifier.type === 'ImportSpecifier') {
    if (specifier.importKind === 'type') continue
    // named import
    mapping.push([
      s.sliceNode(specifier.imported),
      specifier.local.name,
    ])
  } else {
    // default import
    defaultId = specifier.local.name
  }
}

let requireCode: string
if (isBuiltinModule) {
  requireCode =
    source === 'process' || source === 'node:process'
      ? 'globalThis.process'
      : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
  requireCode = `__cjs_require(${JSON.stringify(source)})`
  usingRequire = true
```

```
          }

          const codes: string[] = []
          if (namespaceId) {
            defaultId ||= `_cjs_${namespaceId}_default`
          }
          if (defaultId) {
            // const name = require('cjs-module')
            codes.push(`const ${defaultId} = ${requireCode};`)
          }
          if (namespaceId) {
            // const ns = { ...default, default }
            codes.push(
              `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
            )
          }
          if (mapping.length > 0) {
            codes.push(
              `const {\n${mapping
                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                .join(',\n')}\n} = ${defaultId || requireCode};`,
            )
          }
          s.overwriteNode(stmt, codes.join('\n'))
        }
      }

      if (usingRequire) {
        const preamble = builtinNodeModules
          ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
          : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

        if (code[0] === '#') {
          // skip shebang line
          const firstNewLineIndex = code.indexOf('\n') + 1
          s.appendLeft(firstNewLineIndex, preamble)
        } else {
          s.prepend(preamble)
        }
      }

      return generateTransform(s, fileName)
```

```
            },
        },
    }
}

export async function isPureCJS(
    id: string,
    importer: string,
): Promise<boolean> {
    if (!initted) {
        await init()
    }

    // ignore Node.js built-in modules, as their performance is comparable
    if (id.startsWith('node:')) return false

    try {
        const importResolved = resolvePathSync(id, { url: importer })
        const requireResolved = require.resolve(id, { paths: [importer] })

        // different resolution, respect to original behavior
        if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
            return false
        }

        if (importResolved.endsWith('.cjs')) {
            return true
        } else if (importResolved.endsWith('.js')) {
            const pkgJsonPath = up({ cwd: importResolved })
            if (pkgJsonPath) {
                const pkgType = await getPackageType(pkgJsonPath)
                if (pkgType === 'module') return false
                if (pkgType === 'commonjs') return true
            }

            // detect by parsing
            const contents = await readFile(importResolved, 'utf8')
            try {
                parse(contents, importResolved)
                return true
            } catch {}
        }
    } catch {}
    return false
```

```
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
  const { include, exclude, order, shouldTransform, builtinNodeModules } =
    resolveOptions(userOptions)
  const filter = createFilter(include, exclude)
  let cwd: string
  return {
    name: 'rolldown-plugin-require-cjs',
    async buildStart() {
      if (!initted) {
        await init()
        initted = true
      }
    },
    options(options) {
      if (options.platform !== 'node') {
        this.error(
          '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
```

```
Please make sure to set `platform: "node"` in the options.',
        )
      }
      cwd = options.cwd || process.cwd()
    },
    outputOptions(options) {
      if (!['es', 'esm', 'module'].includes(options.format as any)) {
        throw new Error(
          '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
        )
      }
    },
    renderChunk: {
      order,
      async handler(code, { fileName }, { file, dir }) {
        if (!filter(fileName)) return

        const { body } = parseAst(code, { lang: undefined }, fileName)
        const s = new MagicStringAST(code)
        let usingRequire = false

        for (const stmt of body) {
          if (stmt.type === 'ImportDeclaration') {
            if (stmt.importKind === 'type') continue

            const source = stmt.source.value

            const isBuiltinModule = builtinNodeModules && isBuiltin(source)

            const distFilename =
              file || (dir ? path.join(dir, fileName) : fileName)
            const importer = cwd
              ? path.resolve(cwd, distFilename)
              : distFilename
            const shouldProcess =
              isBuiltinModule ||
              ((await shouldTransform?.(source, importer)) ??
                (await isPureCJS(source, importer)))

            if (!shouldProcess) continue

            if (stmt.specifiers.length === 0) {
              // import 'cjs-module'
              if (isBuiltinModule) {
```

```
        // side-effect free
        s.removeNode(stmt)
    } else {
        // require('cjs-module')
        s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
        usingRequire = true
    }
    continue
}

const mapping: [string, string][] = []
let namespaceId: string | undefined
let defaultId: string | undefined
for (const specifier of stmt.specifiers) {
    // namespace
    if (specifier.type === 'ImportNamespaceSpecifier') {
        // import * as name from 'cjs-module'
        namespaceId = specifier.local.name
    } else if (specifier.type === 'ImportSpecifier') {
        if (specifier.importKind === 'type') continue
        // named import
        mapping.push([
            s.sliceNode(specifier.imported),
            specifier.local.name,
        ])
    } else {
        // default import
        defaultId = specifier.local.name
    }
}

let requireCode: string
if (isBuiltinModule) {
    requireCode =
        source === 'process' || source === 'node:process'
            ? 'globalThis.process'
            : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
} else {
    requireCode = `__cjs_require(${JSON.stringify(source)})`
    usingRequire = true
}

const codes: string[] = []
if (namespaceId) {
```

```
                        defaultId ||= `_cjs_${namespaceId}_default`
                    }
                    if (defaultId) {
                        // const name = require('cjs-module')
                        codes.push(`const ${defaultId} = ${requireCode};`)
                    }
                    if (namespaceId) {
                        // const ns = { ...default, default }
                        codes.push(
                            `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                        )
                    }
                    if (mapping.length > 0) {
                        codes.push(
                            `const {\n${mapping
                                .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                                .join(',\n')}\n} = ${defaultId || requireCode};`,
                        )
                    }
                    s.overwriteNode(stmt, codes.join('\n'))
                }
            }

            if (usingRequire) {
                const preamble = builtinNodeModules
                    ?                              `const                              ${REQUIRE}                              =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
                    : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

                if (code[0] === '#') {
                    // skip shebang line
                    const firstNewLineIndex = code.indexOf('\n') + 1
                    s.appendLeft(firstNewLineIndex, preamble)
                } else {
                    s.prepend(preamble)
                }
            }

            return generateTransform(s, fileName)
        },
    },
  }
}
```

```
export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
```

```
    try {
        const pkg = JSON.parse(contents)
        return pkg.type as string | undefined
    } catch {}
}import { readFile } from 'node:fs/promises'
import { isBuiltin } from 'node:module'
import path from 'node:path'
import process from 'node:process'
import { init, parse } from 'cjs-module-lexer'
import { up } from 'empathic/package'
import { generateTransform, MagicStringAST } from 'magic-string-ast'
import { resolvePathSync } from 'mlly'
import { parseAst } from 'rolldown/parseAst'
import { createFilter } from 'unplugin-utils'
import { resolveOptions, type Options } from './options'
import type { Plugin } from 'rolldown'

export * from './options'

let initted = false

const REQUIRE = `__cjs_require`

export function RequireCJS(userOptions: Options = {}): Plugin {
    const { include, exclude, order, shouldTransform, builtinNodeModules } =
        resolveOptions(userOptions)
    const filter = createFilter(include, exclude)
    let cwd: string
    return {
        name: 'rolldown-plugin-require-cjs',
        async buildStart() {
            if (!initted) {
                await init()
                initted = true
            }
        },
        options(options) {
            if (options.platform !== 'node') {
                this.error(
                    '`rolldown-plugin-require-cjs` plugin is designed only for the Node.js environment.
Please make sure to set `platform: "node"` in the options.',
                )
            }
            cwd = options.cwd || process.cwd()
```

```
},
outputOptions(options) {
  if (!['es', 'esm', 'module'].includes(options.format as any)) {
    throw new Error(
      '`rolldown-plugin-require-cjs` plugin is only necessary for ESM output',
    )
  }
},
renderChunk: {
  order,
  async handler(code, { fileName }, { file, dir }) {
    if (!filter(fileName)) return

    const { body } = parseAst(code, { lang: undefined }, fileName)
    const s = new MagicStringAST(code)
    let usingRequire = false

    for (const stmt of body) {
      if (stmt.type === 'ImportDeclaration') {
        if (stmt.importKind === 'type') continue

        const source = stmt.source.value

        const isBuiltinModule = builtinNodeModules && isBuiltin(source)

        const distFilename =
          file || (dir ? path.join(dir, fileName) : fileName)
        const importer = cwd
          ? path.resolve(cwd, distFilename)
          : distFilename
        const shouldProcess =
          isBuiltinModule ||
          ((await shouldTransform?.(source, importer)) ??
            (await isPureCJS(source, importer)))

        if (!shouldProcess) continue

        if (stmt.specifiers.length === 0) {
          // import 'cjs-module'
          if (isBuiltinModule) {
            // side-effect free
            s.removeNode(stmt)
          } else {
            // require('cjs-module')
            s.overwriteNode(stmt, `${REQUIRE}(${JSON.stringify(source)});`)
            usingRequire = true
          }
          continue
        }

        const mapping: [string, string][] = []
        let namespaceId: string | undefined
        let defaultId: string | undefined
        for (const specifier of stmt.specifiers) {
          // namespace
          if (specifier.type === 'ImportNamespaceSpecifier') {
            // import * as name from 'cjs-module'
            namespaceId = specifier.local.name
          } else if (specifier.type === 'ImportSpecifier') {
            if (specifier.importKind === 'type') continue
            // named import
            mapping.push([
              s.sliceNode(specifier.imported),
              specifier.local.name,
            ])
          } else {
            // default import
            defaultId = specifier.local.name
          }
        }

        let requireCode: string
        if (isBuiltinModule) {
          requireCode =
            source === 'process' || source === 'node:process'
              ? 'globalThis.process'
              : `globalThis.process.getBuiltinModule(${JSON.stringify(source)})`
        } else {
          requireCode = `__cjs_require(${JSON.stringify(source)})`
          usingRequire = true
        }

        const codes: string[] = []
        if (namespaceId) {
          defaultId ||= `_cjs_${namespaceId}_default`
        }
        if (defaultId) {
          // const name = require('cjs-module')
```

```
                codes.push(`const ${defaultId} = ${requireCode};`)
              }
              if (namespaceId) {
                // const ns = { ...default, default }
                codes.push(
                  `const ${namespaceId} = { ...${defaultId}, default: ${defaultId} };`,
                )
              }
              if (mapping.length > 0) {
                codes.push(
                  `const {\n${mapping
                    .map(([k, v]) => `    ${k === v ? v : `${k}: ${v}`}`)
                    .join(',\n')}\n} = ${defaultId || requireCode};`,
                )
              }
              s.overwriteNode(stmt, codes.join('\n'))
            }
          }

          if (usingRequire) {
            const preamble = builtinNodeModules
              ?                    `const                    ${REQUIRE}                    =
globalThis.process.getBuiltinModule("module").createRequire(import.meta.url);\n`
              : `import { createRequire as __cjs_createRequire } from "node:module";
const ${REQUIRE} = __cjs_createRequire(import.meta.url);\n`

            if (code[0] === '#') {
              // skip shebang line
              const firstNewLineIndex = code.indexOf('\n') + 1
              s.appendLeft(firstNewLineIndex, preamble)
            } else {
              s.prepend(preamble)
            }
          }

          return generateTransform(s, fileName)
        },
      },
    }
}

export async function isPureCJS(
  id: string,
  importer: string,
): Promise<boolean> {
  if (!initted) {
    await init()
  }

  // ignore Node.js built-in modules, as their performance is comparable
  if (id.startsWith('node:')) return false

  try {
    const importResolved = resolvePathSync(id, { url: importer })
    const requireResolved = require.resolve(id, { paths: [importer] })

    // different resolution, respect to original behavior
    if (path.resolve(importResolved) !== path.resolve(requireResolved)) {
      return false
    }

    if (importResolved.endsWith('.cjs')) {
      return true
    } else if (importResolved.endsWith('.js')) {
      const pkgJsonPath = up({ cwd: importResolved })
      if (pkgJsonPath) {
        const pkgType = await getPackageType(pkgJsonPath)
        if (pkgType === 'module') return false
        if (pkgType === 'commonjs') return true
      }

      // detect by parsing
      const contents = await readFile(importResolved, 'utf8')
      try {
        parse(contents, importResolved)
        return true
      } catch {}
    }
  } catch {}
  return false
}

async function getPackageType(path: string): Promise<string | undefined> {
  const contents = await readFile(path, 'utf8')
  try {
    const pkg = JSON.parse(contents)
    return pkg.type as string | undefined
  } catch {}
```

```
}
```