link: null
title: 珠峰架构师成长计划
description: bin\vite3.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=114 sentences=785, words=6687

1.核心知识

```
npm install connect es-module-lexer resolve check-is-array esbuild fast-glob fs-extra serve-static magic-string chokidar ws --save
```

- Connect (https://www.npmjs.com/package/connect)是一个框架，它使用被称为中间件的模块化组件，以可重用的方式实现web程序的逻辑
- 在Connect中，中间件组件是一个函数，它拦截HTTP服务器提供的请求和响应，执行逻辑，然后，或者结束响应，或者把它传递给下一个中间件组件
- Connect用分配器把中间件 `&#x8FDE;&#x63A5;`在一起
- Express构建在Connect之上的更高层的框架

```
const connect = require('connect');
const http = require('http');

const middlewares = connect();
middlewares.use(function (req, res, next) {
  console.log('middleware1');
  next();
});
middlewares.use(function (req, res, next) {
  console.log('middleware2');
  next();
});
middlewares.use(function (req, res, next) {
  res.end('Hello from Connect!');
});
http.createServer(middlewares).listen(3000);
```

```
const connect = require('connect');
const static = require('serve-static');
const http = require('http');

const middlewares = connect();
middlewares.use(static(__dirname));
http.createServer(middlewares).listen(3001);
```

```
const { init, parse } = require('es-module-lexer');
(async () => {
  await init;
  const [imports, exports] = parse(`import _ from 'lodash';\nexport var p = 5`);
  console.log(imports);
  console.log(exports);
})();
```

```
const resolve = require('resolve');
const res = resolve.sync('check-is-array', { basedir: __dirname });
console.log(res);
```

```
const fg = require('fast-glob');
(async () => {
  const entries = await fg(['**/*.js']);
  console.log(entries);
})();
```

```
const MagicString = require('magic-string');
const ms = new MagicString('var age = 10');
ms.overwrite(10, 12, '11');
console.log(ms.toString());
```

```
esbuild index.js
esbuild index.js --outfile=dist.js
esbuild index.js --outfile=dist.js --bundle
esbuild index.js --outfile=dist.js --bundle --target=esnext
esbuild index.js --outfile=dist.js --bundle --target=esnext --platform=node
esbuild index.js --outfile=dist.js  --platform=node --format=esm
esbuild index.js --outfile=dist.js  --platform=node --format=esm --watch
esbuild index.js --outfile=dist.js  --platform=node --format=esm --define:AGE=12
esbuild index.js --outfile=dist.js  --bundle --platform=browser --format=iife --loader:.jpg=dataurl
esbuild index.js --outdir=dist
esbuild index.jsx --outdir=dist
esbuild index.jsx --outdir=dist --bundle --jsx-factory=createVnode
```

```
const path = require('path');
const { build } = require('esbuild');
(async function () {
  await build({
    absWorkingDir: process.cwd(),
    entryPoints: [path.resolve('main.js')],
    outfile: path.resolve('dist/main.js'),
    bundle: true,
    write: true,
    format: 'esm'
  })
})();
```

```
let envPlugin = {
  name: 'env',
  setup(build) {

    build.onResolve({ filter: /^env$/ }, ({ path }) => ({
      path,
      namespace: 'env-ns',
    }))

    build.onLoad({ filter: /.*/, namespace: 'env-ns' }, () => ({
      contents: JSON.stringify(process.env),
      loader: 'json',
    }))
  },
}
require('esbuild').build({
  entryPoints: ['app.js'],
  bundle: true,
  outfile: 'out.js',
  plugins: [envPlugin],
}).catch(() => process.exit(1))
```

2.实现命令行

```
{
  "bin": {
    "vite3": "./bin/vite3.js"
  },
}
```

bin\vite3.js

```

require('../lib/cli');
```

lib\cli.js

```
console.log('vite3');
```

3.实现http服务器

lib\cli.js

```
+let { createServer } = require('./server');
+(async function () {
+  const server = await createServer();
+  server.listen(9999);
+})();
```

lib\server\index.js

```
const connect = require('connect');
async function createServer() {
  const middlewares = connect();
  const server = {
    async listen(port) {
      require('http').createServer(middlewares)
        .listen(port, async () => {
          console.log(`dev server running at: http://localhost:${port}`)
        })
    }
  }
  return server;
}
exports.createServer = createServer;
```

4.实现静态文件中间件

lib\server\index.js

```
const connect = require('connect');
+const serveStaticMiddleware = require('./middlewares/static');
+const resolveConfig = require('../config');
async function createServer() {
+ const config = await resolveConfig()
  const middlewares = connect();
  const server = {
    async listen(port) {
      require('http').createServer(middlewares)
        .listen(port, async () => {
          console.log(`dev server running at: http://localhost:${port}`)
        })
    }
  }
+ middlewares.use(serveStaticMiddleware(config))
  return server;
}
exports.createServer = createServer;
```

lib\server\middlewares\static.js

```
const static = require('serve-static');
function serveStaticMiddleware({ root }) {
  return static(root)
}
module.exports = serveStaticMiddleware;
```

lib\config.js

```
const { normalizePath } = require('./utils');
async function resolveConfig() {
  const root = normalizePath(process.cwd());
  let config = {
    root
  };
  return config;
}
module.exports = resolveConfig;
```

lib\utils.js

```
function normalizePath(id) {
  return id.replace(/\\/g, '/')
}
exports.normalizePath = normalizePath;
```

index.html

```

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vite2title>
head>

<body>
  <div id="app">div>
  <script src="/src/main.js" type="module">script>
body>

html>
```

src\main.js

```
console.log('main');
```

5.分析第三方依赖

lib\server\index.js

```
const connect = require('connect');
const serveStaticMiddleware = require('./middlewares/static');
const resolveConfig = require('../config');
+const { createOptimizeDepsRun } = require('../optimizer');
async function createServer() {
  const config = await resolveConfig()
  const middlewares = connect();
  const server = {
    async listen(port) {
+     await runOptimize(config, server)
      require('http').createServer(middlewares)
        .listen(port, async () => {
          console.log(`dev server running at: http://localhost:${port}`)
        })
    }
  }
  middlewares.use(serveStaticMiddleware(config))
  return server;
}
+async function runOptimize(config, server) {
+  await createOptimizeDepsRun(config)
+}
exports.createServer = createServer;
```

lib\optimizer\index.js

```
const scanImports = require('./scan');
async function createOptimizeDepsRun(config) {
  const deps = await scanImports(config)
  console.log(deps);
}
exports.createOptimizeDepsRun = createOptimizeDepsRun;
```

lib\optimizer\scan.js

```
const { build } = require('esbuild');
const esbuildScanPlugin = require('./esbuildScanPlugin');
const path = require('path');
async function scanImports(config) {
  const depImports = {};
  const esPlugin = await esbuildScanPlugin(config, depImports);
  await build({
    absWorkingDir: config.root,
    entryPoints: [path.resolve('./index.html')],
    bundle: true,
    format: 'esm',
    outfile: 'dist/index.js',
    write: true,
    plugins: [esPlugin]
  })
  return depImports;
}
module.exports = scanImports;
```

lib\optimizer\esbuildScanPlugin.js

```
const fs = require('fs-extra');
const path = require('path');
const { createPluginContainer } = require('../server/pluginContainer');
const resolvePlugin = require('../plugins/resolve');
const { normalizePath } = require('../utils');
const htmlTypesRE = /\.html$/
const scriptModuleRE = /<\/script>/</span>;
<span class="hljs-keyword">const</span> JS_TYPES_RE = <span class="hljs-regexp">/\.js$/</span>;
<span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">esbuildScanPlugin</span>
(<span class="hljs-params">config, depImports</span>) </span>{
  config.plugins = [resolvePlugin(config)];
  <span class="hljs-keyword">const</span> container = <span class="hljs-keyword">await</span> createPluginContainer(config)
  <span class="hljs-keyword">const</span> resolve = <span class="hljs-keyword">async</span> (id, importer) => {
    <span class="hljs-keyword">return</span> <span class="hljs-keyword">await</span> container.resolveId(id, importer)
  }
  <span class="hljs-keyword">return</span> {
    <span class="hljs-attr">name</span>: <span class="hljs-string">'vite:dep-scan'</span>,
    setup(build) {
      build.onResolve({ <span class="hljs-attr">filter</span>: htmlTypesRE }, <span class="hljs-keyword">async</span> ({ path, importer }) => {
        <span class="hljs-keyword">const</span> resolved = <span class="hljs-keyword">await</span> resolve(path, importer)
        <span class="hljs-keyword">if</span> (resolved) {
          <span class="hljs-keyword">return</span> {
            <span class="hljs-attr">path</span>: resolved.id || resolved,
            <span class="hljs-attr">namespace</span>: <span class="hljs-string">'html'</span>
          }
        }
      })
      build.onResolve({ <span class="hljs-attr">filter</span>: <span class="hljs-regexp">/.*/</span> }, <span class="hljs-keyword">async</span> ({ path,
importer }) => {
        <span class="hljs-keyword">const</span> resolved = <span class="hljs-keyword">await</span> resolve(path, importer)
        <span class="hljs-keyword">if</span> (resolved) {
          <span class="hljs-keyword">const</span> id = resolved.id || resolved;
          <span class="hljs-keyword">const</span> included = id.includes(<span class="hljs-string">'node_modules'</span>);
          <span class="hljs-keyword">if</span> (included) {
            depImports[path] = normalizePath(id)
            <span class="hljs-keyword">return</span> {
              <span class="hljs-attr">path</span>: id,
              <span class="hljs-attr">external</span>: <span class="hljs-literal">true</span>
            }
          }
          <span class="hljs-keyword">return</span> {
            <span class="hljs-attr">path</span>: id
          }
        }
        <span class="hljs-keyword">return</span> { path }
      })
      build.onLoad({ <span class="hljs-attr">filter</span>: htmlTypesRE, <span class="hljs-attr">namespace</span>: <span class="hljs-string">'html'</span> },
<span class="hljs-keyword">async</span> ({ path }) => {
        <span class="hljs-keyword">let</span> html = fs.readFileSync(path, <span class="hljs-string">'utf-8'</span>)
        <span class="hljs-keyword">let</span> [, scriptSrc] = html.match(scriptModuleRE);
        <span class="hljs-keyword">let</span> js = <span class="hljs-string">`import <span class="hljs-subst">${<span class="hljs-
built_in">JSON</span>.stringify(scriptSrc)}</span>;\n`</span>
        <span class="hljs-keyword">return</span> {
          <span class="hljs-attr">loader</span>: <span class="hljs-string">'js'</span>,
          <span class="hljs-attr">contents</span>: js
        }
      })
      build.onLoad({ <span class="hljs-attr">filter</span>: JS_TYPES_RE }, ({ <span class="hljs-attr">path</span>: id }) => {
        <span class="hljs-keyword">let</span> ext = path.extname(id).slice(<span class="hljs-number">1</span>)
        <span class="hljs-keyword">let</span> contents = fs.readFileSync(id, <span class="hljs-string">'utf-8'</span>)
        <span class="hljs-keyword">return</span> {
          <span class="hljs-attr">loader</span>: ext,
          contents
        }
      })
    }
  }
}
<span class="hljs-built_in">module</span>.exports = esbuildScanPlugin;
</code></pre><p>lib\server\pluginContainer.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> { normalizePath } = <span class="hljs-
built_in">require</span>(<span class="hljs-string">"../utils"</span>);
<span class="hljs-keyword">const</span> path = <span class="hljs-built_in">require</span>(<span class="hljs-string">'path'</span>);
<span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-
title">createPluginContainer</span>(<span class="hljs-params">{ plugins,root }</span>) </span>{
  <span class="hljs-class"><span class="hljs-keyword">class</span> <span class="hljs-title">PluginContext</span> </span>{
    <span class="hljs-keyword">async</span> resolve(id, , importer = path.join(root, <span class="hljs-string">'index.html'</span>)) {
      <span class="hljs-keyword">return</span> <span class="hljs-keyword">await</span> container.resolveId(id, importer)
    }
  }
  <span class="hljs-keyword">const</span> container = {
     <span class="hljs-keyword">async</span> resolveId(id, importer) {
      <span class="hljs-keyword">let</span> ctx = <span class="hljs-keyword">new</span> PluginContext();
      <span class="hljs-keyword">let</span> resolveId = id;
      <span class="hljs-keyword">for</span> (<span class="hljs-keyword">const</span> plugin <span class="hljs-keyword">of</span> plugins) {
        <span class="hljs-keyword">if</span> (!plugin.resolveId) <span class="hljs-keyword">continue</span>;
        <span class="hljs-keyword">const</span> result = <span class="hljs-keyword">await</span> plugin.resolveId.call(ctx, id, importer);
        <span class="hljs-keyword">if</span> (result) {
          resolveId = result.id || result;
          <span class="hljs-keyword">break</span>;
        }
      }
      <span class="hljs-keyword">return</span> { <span class="hljs-attr">id</span>: normalizePath(resolveId) }
    }
  }
  <span class="hljs-keyword">return</span> container;
}
exports.createPluginContainer = createPluginContainer;
</code></pre><p>lib\plugins\resolve.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> fs = <span class="hljs-built_in">require</span>
(<span class="hljs-string">'fs'</span>);
<span class="hljs-keyword">const</span> path = <span class="hljs-built_in">require</span>(<span class="hljs-string">'path'</span>);
<span class="hljs-keyword">const</span> resolve = <span class="hljs-built_in">require</span>(<span class="hljs-string">'resolve'</span>);
<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">resolvePlugin</span>(<span class="hljs-params">config</span>)
</span>{
  <span class="hljs-keyword">return</span> {
    <span class="hljs-attr">name</span>: <span class="hljs-string">'vite:resolve'</span>,
    resolveId(id, importer) {

      <span class="hljs-keyword">if</span> (id.startsWith(<span class="hljs-string">'/'</span>)) {
        <span class="hljs-keyword">return</span> { <span class="hljs-attr">id</span>: path.resolve(config.root, id.slice(<span class="hljs-number">1</span>)) };
      }
      <span class="hljs-keyword">if</span> (path.isAbsolute(id)) {
        <span class="hljs-keyword">return</span> { id }
      }

      <span class="hljs-keyword">if</span> (id.startsWith(<span class="hljs-string">'.'</span>)) {
        <span class="hljs-keyword">const</span> basedir = path.dirname(importer);
        <span class="hljs-keyword">const</span> fsPath = path.resolve(basedir, id)
        <span class="hljs-keyword">return</span> { <span class="hljs-attr">id</span>: fsPath };
      }

      <span class="hljs-keyword">let</span> res = tryNodeResolve(id, importer, config);
      <span class="hljs-keyword">if</span> (res) {
        <span class="hljs-keyword">return</span> res;
      }
    }
  }
}

<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">tryNodeResolve</span>(<span class="hljs-params">id, importer,
config</span>) </span>{
  <span class="hljs-keyword">const</span> pkgPath = resolve.sync(<span class="hljs-string">`<span class="hljs-subst">${id}</span>/package.json`</span>, { <span
class="hljs-attr">basedir</span>: config.root })
  <span class="hljs-keyword">const</span> pkgDir = path.dirname(pkgPath)
  <span class="hljs-keyword">const</span> pkg = <span class="hljs-built_in">JSON</span>.parse(fs.readFileSync(pkgPath, <span class="hljs-string">'utf-
8'</span>))
  <span class="hljs-keyword">const</span> entryPoint = pkg.module
  <span class="hljs-keyword">const</span> entryPointPath = path.join(pkgDir, entryPoint)
  <span class="hljs-keyword">return</span> { <span class="hljs-attr">id</span>: entryPointPath }
}
<span class="hljs-built_in">module</span>.exports = resolvePlugin;
</code></pre><p>lib\server\index.js</p><pre><code class="lang-diff">const connect = require('connect');
const http = require('http');
const serveStaticMiddleware = require('./middlewares/static');
const resolveConfig = require('../config');
const { createOptimizeDepsRun } = require('../optimizer');
async function createServer() {
  const config = await resolveConfig();
  const middlewares = connect();
  const server = {
    async listen(port) {
<span class="hljs-addition">+     await runOptimize(config, server)</span>
      http.createServer(middlewares).listen(port, async () => {
        console.log(`server running at http://localhost:${port}`);
      });
    }
  }
  middlewares.use(serveStaticMiddleware(config));
  return server;
}
<span class="hljs-addition">+async function runOptimize(config, server) {</span>
<span class="hljs-addition">+  const optimizeDeps = await createOptimizeDepsRun(config);</span>
<span class="hljs-addition">+  server._optimizeDepsMetadata = optimizeDeps.metadata</span>
}
exports.createServer = createServer;
</code></pre><p>lib\config.js</p><pre><code class="lang-diff"><span class="hljs-addition">+const path = require('path');</span>
const { normalizePath } = require('./utils');
async function resolveConfig() {
  //当前的根目录 window \\  linux /
  const root = normalizePath(process.cwd());
<span class="hljs-addition">+ const cacheDir = normalizePath(path.resolve(`node_modules/.vite7`))</span>
  let config = {
    root,
<span class="hljs-addition">+   cacheDir</span>
  }
  return config;
}
module.exports = resolveConfig;
</code></pre><p>lib\utils.js</p><pre><code class="lang-js"><span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-
title">normalizePath</span>(<span class="hljs-params">id</span>) </span>{
  <span class="hljs-keyword">return</span> id.replace(<span class="hljs-regexp">/\\/g</span>, <span class="hljs-string">'/'</span>)
}
exports.normalizePath = normalizePath;
</code></pre><p>lib\optimizer\index.js</p><pre><code class="lang-diff">const scanImports = require('./scan');
<span class="hljs-addition">+const fs = require('fs-extra');</span>
<span class="hljs-addition">+const path = require('path');</span>
<span class="hljs-addition">+const { build } = require('esbuild');</span>
<span class="hljs-addition">+const { normalizePath } = require('../utils');</span>
async function createOptimizeDepsRun(config) {
  const deps = await scanImports(config);
<span class="hljs-addition">+ const { cacheDir } = config;</span>
<span class="hljs-addition">+ const depsCacheDir = path.resolve(cacheDir, 'deps')</span>
<span class="hljs-addition">+ const metadataPath = path.join(depsCacheDir, '_metadata.json');</span>
<span class="hljs-addition">+ const metadata = {</span>
<span class="hljs-addition">+   optimized: {}</span>
<span class="hljs-addition">+ }</span>
<span class="hljs-addition">+ for (const id in deps) {</span>
<span class="hljs-addition">+   const entry = deps[id]</span>
<span class="hljs-addition">+   metadata.optimized[id] = {</span>
<span class="hljs-addition">+     file: normalizePath(path.resolve(depsCacheDir, id + '.js')),</span>
<span class="hljs-addition">+     src: entry</span>
<span class="hljs-addition">+   }</span>
<span class="hljs-addition">+   await build({</span>
<span class="hljs-addition">+     absWorkingDir: process.cwd(),</span>
<span class="hljs-addition">+     entryPoints: [deps[id]],</span>
<span class="hljs-addition">+     outfile: path.resolve(depsCacheDir, id + '.js'),</span>
<span class="hljs-addition">+     bundle: true,</span>
<span class="hljs-addition">+     write: true,</span>
<span class="hljs-addition">+     format: 'esm'</span>
<span class="hljs-addition">+   })</span>
<span class="hljs-addition">+ }</span>
<span class="hljs-addition">+ await fs.ensureDir(depsCacheDir);</span>
<span class="hljs-addition">+ await fs.writeFile(metadataPath, JSON.stringify(metadata, (key, value) => {</span>
<span class="hljs-addition">+   if (key === 'file' || key === 'src') {</span>
<span class="hljs-addition">+     console.log(depsCacheDir, value);</span>
<span class="hljs-addition">+     return normalizePath(path.relative(depsCacheDir, value));</span>
<span class="hljs-addition">+   }</span>
<span class="hljs-addition">+   return value</span>
<span class="hljs-addition">+ }, 2));</span>
<span class="hljs-addition">+ return { metadata };</span>
}
exports.createOptimizeDepsRun = createOptimizeDepsRun;
</code></pre><h2 id="t397.修改导入路径">7.修改导入路径 </h2><ul><li><code>import { createApp } from 'vue'</code></li><li><code>import { createApp } from
'/node_modules/.vite7/deps/vue.js'</code></li></ul><p>lib\server\index.js</p><pre><code class="lang-diff">const connect = require('connect');
const http = require('http');
const serveStaticMiddleware = require('./middlewares/static');
const resolveConfig = require('../config');
const { createOptimizeDepsRun } = require('../optimizer');
<span class="hljs-addition">+const transformMiddleware = require('./middlewares/transform');</span>
<span class="hljs-addition">+const { createPluginContainer } = require('./pluginContainer');</span>
async function createServer() {
  const config = await resolveConfig();
  const middlewares = connect();
<span class="hljs-addition">+ const pluginContainer = await createPluginContainer(config)</span>
  const server = {
<span class="hljs-addition">+   pluginContainer,</span>
    async listen(port) {
      await runOptimize(config, server)
      http.createServer(middlewares).listen(port, async () => {
        console.log(`server running at http://localhost:${port}`);
      });
    }
  }
<span class="hljs-addition">+ for (const plugin of config.plugins) {</span>
<span class="hljs-addition">+   if (plugin.configureServer) {</span>
<span class="hljs-addition">+     await plugin.configureServer(server)</span>
<span class="hljs-addition">+   }</span>
<span class="hljs-addition">+ }</span>
<span class="hljs-addition">+ middlewares.use(transformMiddleware(server))</span>
  middlewares.use(serveStaticMiddleware(config));
  return server;
}
async function runOptimize(config, server) {
  const optimizeDeps = await createOptimizeDepsRun(config);
  server._optimizeDepsMetadata = optimizeDeps.metadata
}
exports.createServer = createServer;
</code></pre><p>lib\server\middlewares\transform.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> { isJSRequest } = <span class="hljs-
built_in">require</span>(<span class="hljs-string">'../../utils'</span>);
<span class="hljs-keyword">const</span> send = <span class="hljs-built_in">require</span>(<span class="hljs-string">'../send'</span>);
<span class="hljs-keyword">const</span> transformRequest = <span class="hljs-built_in">require</span>(<span class="hljs-string">'../transformRequest'</span>);
<span class="hljs-keyword">const</span> { parse } = <span class="hljs-built_in">require</span>(<span class="hljs-string">'url'</span>);
<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">transformMiddleware</span>(<span class="hljs-
params">server</span>) </span>{
  <span class="hljs-keyword">return</span> <span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> (<span
class="hljs-params">req, res, next</span>) </span>{
    <span class="hljs-keyword">if</span> (req.method !== <span class="hljs-string">'GET'</span>) {
      <span class="hljs-keyword">return</span> next()
    }
    <span class="hljs-keyword">let</span> url = parse(req.url).pathname;
    <span class="hljs-keyword">if</span> (isJSRequest(url)) {
      <span class="hljs-keyword">const</span> result = <span class="hljs-keyword">await</span> transformRequest(url, server)
      <span class="hljs-keyword">if</span> (result) {
        <span class="hljs-keyword">const</span> type = <span class="hljs-string">'js'</span>
        <span class="hljs-keyword">return</span> send(req, res, result.code, type)
      }
    } <span class="hljs-keyword">else</span> {
      <span class="hljs-keyword">return</span> next();
    }
  }
}
<span class="hljs-built_in">module</span>.exports = transformMiddleware
</code></pre><p>lib\server\pluginContainer.js</p><pre><code class="lang-diff">const { normalizePath } = require("../utils");
const path = require('path');
async function createPluginContainer({ plugins,root }) {
  class PluginContext {
<span class="hljs-addition">+   async resolve(id, importer= path.join(root, 'index.html')) {</span>
<span class="hljs-addition">+     return await container.resolveId(id, importer)</span>
<span class="hljs-addition">+   }</span>
  }
  //插件容器是一个用来执行插件的容器
  const container = {
    //resolve是一个方法，是一个根据标记符计算路径的方法
    //vue=>vue在硬盘上对应路径
    async resolveId(id, importer) {
      let ctx = new PluginContext();
      let resolveId = id;
      for (const plugin of plugins) {
        if (!plugin.resolveId) continue;
        const result = await plugin.resolveId.call(ctx, id, importer);
        if (result) {
          resolveId = result.id || result;
          break;
        }
      }
      return { id: normalizePath(resolveId) }
    },
<span class="hljs-addition">+   async load(id) {</span>
<span class="hljs-addition">+     const ctx = new PluginContext()</span>
<span class="hljs-addition">+     for (const plugin of plugins) {</span>
<span class="hljs-addition">+       if (!plugin.load) continue</span>
<span class="hljs-addition">+       const result = await plugin.load.call(ctx, id)</span>
<span class="hljs-addition">+       if (result !== null) {</span>
<span class="hljs-addition">+         return result</span>
<span class="hljs-addition">+       }</span>
<span class="hljs-addition">+     }</span>
<span class="hljs-addition">+     return null</span>
<span class="hljs-addition">+   },</span>
```

```
<span class="hljs-addition">+   async transform(code, id) {</span>
<span class="hljs-addition">+     for (const plugin of plugins) {</span>
<span class="hljs-addition">+       if (!plugin.transform) continue</span>
<span class="hljs-addition">+       const ctx = new PluginContext()</span>
<span class="hljs-addition">+       const result = await plugin.transform.call(ctx, code, id)</span>
<span class="hljs-addition">+       if (!result) continue</span>
<span class="hljs-addition">+       code = result.code || result;</span>
<span class="hljs-addition">+     }</span>
<span class="hljs-addition">+     return { code }</span>
<span class="hljs-addition">+   }</span>
  }
  return container;
}
exports.createPluginContainer = createPluginContainer;
</code></pre><p>lib\utils.js</p><pre><code class="lang-diff">function normalizePath(id) {
  return id.replace(/\\/g, '/')
}
exports.normalizePath = normalizePath;

<span class="hljs-addition">+const knownJsSrcRE = /\.js/</span>
<span class="hljs-addition">+const isJSRequest = (url) => {</span>
<span class="hljs-addition">+  if (knownJsSrcRE.test(url)) {</span>
<span class="hljs-addition">+    return true</span>
<span class="hljs-addition">+  }</span>
<span class="hljs-addition">+  return false</span>
<span class="hljs-addition">+}</span>
<span class="hljs-addition">+exports.isJSRequest = isJSRequest;</span>
</code></pre><p>lib\server\transformRequest.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> fs = <span class="hljs-
built_in">require</span>(<span class="hljs-string">'fs-extra'</span>);
<span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">transformRequest</span>
(<span class="hljs-params">url, server</span>) </span>{
  <span class="hljs-keyword">const</span> { pluginContainer } = server
  <span class="hljs-keyword">const</span> { id } = <span class="hljs-keyword">await</span> pluginContainer.resolveId(url);
  <span class="hljs-keyword">const</span> loadResult = <span class="hljs-keyword">await</span> pluginContainer.load(id)
   <span class="hljs-keyword">let</span> code;
  <span class="hljs-keyword">if</span> (loadResult) {
    code = loadResult.code;;
  } <span class="hljs-keyword">else</span> {
    code = <span class="hljs-keyword">await</span> fs.readFile(id, <span class="hljs-string">'utf-8'</span>)
  }
  <span class="hljs-keyword">const</span> transformResult = <span class="hljs-keyword">await</span> pluginContainer.transform(code, id)
  <span class="hljs-keyword">return</span> transformResult;
}
<span class="hljs-built_in">module</span>.exports = transformRequest;
</code></pre><p>lib\server\send.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> alias = {
  <span class="hljs-attr">js</span>: <span class="hljs-string">'application/javascript'</span>,
  <span class="hljs-attr">css</span>: <span class="hljs-string">'text/css'</span>,
  <span class="hljs-attr">html</span>: <span class="hljs-string">'text/html'</span>,
  <span class="hljs-attr">json</span>: <span class="hljs-string">'application/json'</span>
}
<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">send</span>(<span class="hljs-params">_req, res, content,
type</span>) </span>{
  res.setHeader(<span class="hljs-string">'Content-Type'</span>, alias[type] || type)
  res.statusCode = <span class="hljs-number">200</span>
  <span class="hljs-keyword">return</span> res.end(content)
}
<span class="hljs-built_in">module</span>.exports = send;
</code></pre><p>lib\plugins\index.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> importAnalysisPlugin = <span class="hljs-
built_in">require</span>(<span class="hljs-string">'./importAnalysis'</span>);
<span class="hljs-keyword">const</span> preAliasPlugin = <span class="hljs-built_in">require</span>(<span class="hljs-string">'./preAlias'</span>);
<span class="hljs-keyword">const</span> resolvePlugin = <span class="hljs-built_in">require</span>(<span class="hljs-string">'./resolve'</span>);
<span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">resolvePlugins</span>
(<span class="hljs-params">config</span>) </span>{
  <span class="hljs-keyword">return</span> [
    preAliasPlugin(config),
    resolvePlugin(config),
    importAnalysisPlugin(config)
  ]
}
exports.resolvePlugins = resolvePlugins;
</code></pre><p>lib\plugins\importAnalysis.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> { init, parse } = <span class="hljs-
built_in">require</span>(<span class="hljs-string">'es-module-lexer'</span>)
<span class="hljs-keyword">const</span> MagicString = <span class="hljs-built_in">require</span>(<span class="hljs-string">'magic-string'</span>);
<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">importAnalysisPlugin</span>(<span class="hljs-
params">config</span>) </span>{
  <span class="hljs-keyword">const</span> { root } = config
  <span class="hljs-keyword">return</span> {
    <span class="hljs-attr">name</span>: <span class="hljs-string">'vite:import-analysis'</span>,
    <span class="hljs-keyword">async</span> transform(source, importer) {
      <span class="hljs-keyword">await</span> init
      <span class="hljs-keyword">let</span> imports = parse(source)[<span class="hljs-number">0</span>]
      <span class="hljs-keyword">if</span> (!imports.length) {
        <span class="hljs-keyword">return</span> source
      }
      <span class="hljs-keyword">let</span> ms = <span class="hljs-keyword">new</span> MagicString(source);
      <span class="hljs-keyword">const</span> normalizeUrl = <span class="hljs-keyword">async</span> (url) => {
        <span class="hljs-keyword">const</span> resolved = <span class="hljs-keyword">await</span> <span class="hljs-keyword">this</span>.resolve(url, importer)
        <span class="hljs-keyword">if</span> (resolved.id.startsWith(root + <span class="hljs-string">'/'</span>)) {
          url = resolved.id.slice(root.length)
        }
        <span class="hljs-keyword">return</span> url;
      }
      <span class="hljs-keyword">for</span> (<span class="hljs-keyword">let</span> index = <span class="hljs-number">0</span>; index < imports.length; index++)
{
        <span class="hljs-keyword">const</span> { <span class="hljs-attr">s</span>: start, <span class="hljs-attr">e</span>: end, <span class="hljs-
attr">n</span>: specifier } = imports[index]
        <span class="hljs-keyword">if</span> (specifier) {
          <span class="hljs-keyword">const</span> normalizedUrl = <span class="hljs-keyword">await</span> normalizeUrl(specifier)
          <span class="hljs-keyword">if</span> (normalizedUrl !== specifier) {
            ms.overwrite(start, end, normalizedUrl)
          }
        }
      }
      <span class="hljs-keyword">return</span> ms.toString()
    }
  }
}
<span class="hljs-built_in">module</span>.exports = importAnalysisPlugin;
</code></pre><p>lib\plugins\preAlias.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> path = <span class="hljs-built_in">require</span>
(<span class="hljs-string">'path'</span>);
<span class="hljs-keyword">const</span> fs = <span class="hljs-built_in">require</span>(<span class="hljs-string">'fs-extra'</span>);
<span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">preAliasPlugin</span>(<span class="hljs-params"></span>) </span>
{
  <span class="hljs-keyword">let</span> server
  <span class="hljs-keyword">return</span> {
    <span class="hljs-attr">name</span>: <span class="hljs-string">'vite:pre-alias'</span>,
    configureServer(_server) {
      server = _server
    },
    resolveId(id) {
      <span class="hljs-keyword">const</span> metadata = server._optimizeDepsMetadata;
      <span class="hljs-keyword">const</span> isOptimized = metadata.optimized[id]
      <span class="hljs-keyword">if</span> (isOptimized) {
        <span class="hljs-keyword">return</span> {
          <span class="hljs-attr">id</span>: isOptimized.file
        };
      }
    }
  }
}
<span class="hljs-built_in">module</span>.exports = preAliasPlugin;
</code></pre><p>lib\config.js</p><pre><code class="lang-js"><span class="hljs-keyword">const</span> path = <span class="hljs-built_in">require</span>(<span
class="hljs-string">'path'</span>);
<span class="hljs-keyword">const</span> { normalizePath } = <span class="hljs-built_in">require</span>(<span class="hljs-string">'./utils'</span>);
<span class="hljs-keyword">const</span> { resolvePlugins } = <span class="hljs-built_in">require</span>(<span class="hljs-string">'./plugins'</span>);
<span class="hljs-keyword">async</span> <span class="hljs-function"><span class="hljs-keyword">function</span> <span class="hljs-title">resolveConfig</span>
(<span class="hljs-params"></span>) </span>{

  <span class="hljs-keyword">const</span> root = normalizePath(process.cwd());
  <span class="hljs-keyword">const</span> cacheDir = normalizePath(path.resolve(<span class="hljs-string">`node_modules/.vite7`</span>))
  <span class="hljs-keyword">let</span> config = {
    root,
    cacheDir
  }
  <span class="hljs-keyword">const</span> plugins = <span class="hljs-keyword">await</span> resolvePlugins(config);
  config.plugins = plugins;
  <span class="hljs-keyword">return</span> config;
}
<span class="hljs-built_in">module</span>.exports = resolveConfig;
</code></pre><h2 id="t508.支持vue插件">8.支持vue插件 </h2><p>lib\optimizer\esbuildDepPlugin.js</p><pre><code class="lang-diff">const path = require('path');
const fs = require('fs-extra');
const htmlTypesRE = /\.html$/;
const scriptModuleRE = /<script src\="(.+?)" type\="module"><\/script>/;
const { createPluginContainer } = require('../server/pluginContainer');
const resolvePlugin = require('../plugins/resolve');
const jsRE = /\.js$/;
async function esBuildScanPlugin(config, deps) {
  //在此处其实调用的vite插件系统
  config.plugins = [resolvePlugin(config)];
  const container = await createPluginContainer(config);
  const resolve = async (id, importer) => {
    return await container.resolveId(id, importer);
  }
  return {
    name: 'vite:dep-scan',
    setup(build) {
      //X [ERROR] No loader is configured for ".vue" files: src/App.vue
<span class="hljs-addition">+     build.onResolve(</span>
<span class="hljs-addition">+       {</span>
<span class="hljs-addition">+         filter: /\.vue$/</span>
<span class="hljs-addition">+       },</span>
<span class="hljs-addition">+       async ({ path: id, importer }) => {</span>
<span class="hljs-addition">+         const resolved = await resolve(id, importer)</span>
<span class="hljs-addition">+         if (resolved) {</span>
<span class="hljs-addition">+           return {</span>
<span class="hljs-addition">+             path: resolved.id,</span>
<span class="hljs-addition">+             external: true</span>
<span class="hljs-addition">+           }</span>
<span class="hljs-addition">+         }</span>
<span class="hljs-addition">+       }</span>
<span class="hljs-addition">+     )</span>
      //用来处理路径的
      build.onResolve({ filter: htmlTypesRE }, async ({ path, importer }) => {
        //path=C:\aproject\vite5\doc\index.html importer 空
        const resolved = await resolve(path, importer);
        if (resolved) {
          return {
            path: resolved.id || resolved,
            namespace: 'html' //为了更细化区分不同的文件类型，我可以给文件添加一个命名空间
          }
        }
      });
      //对于其它所有的类型文件我们也进行处理
      build.onResolve({ filter: /.*/ }, async ({ path, importer }) => {
        const resolved = await resolve(path, importer);
        //返回值可能是 {id:xx} 或 xx
        //C:\aproject\vite5\doc\main.js
        if (resolved) {
          const id = resolved.id || resolved;
          const included = id.includes('node_modules');
          if (included) {
            //deps.vue = "C:/aproject/viteproject/node_modules/vue/dist/vue.runtime.esm-bundler.js"
            deps[path] = id;
            return {
              path,
              external: true //external设置为true的话说明这是一个外部模块，不会进行后续的打包分析，直接返回了
            }
          }
          return { path: id }
        }
        return { path }
      });
      //用来处理读取内容 自定义读取器
      build.onLoad({ filter: htmlTypesRE, namespace: 'html' }, async ({ path }) => {
        let html = fs.readFileSync(path, 'utf8');
        let [, scriptSrc] = html.match(scriptModuleRE);
        let js = `import ${JSON.stringify(scriptSrc)}`;//import "/main.js"
        return {
          loader: 'js',
          contents: js
        }
      })
      build.onLoad({ filter: jsRE }, async ({ path: id }) => {
        let ext = path.extname(id).slice(1);// .js  js
        const contents = fs.readFileSync(id, 'utf8');
        return {
          loader: ext,
          contents
        }
      })
    }
  }
}
module.exports = esBuildScanPlugin;
</code></pre><p>lib\plugins\index.js</p><pre><code class="lang-diff">const importAnalysisPlugin = require('./importAnalysis');
const preAliasPlugin = require('./preAlias');
const resolvePlugin = require('./resolve');
<span class="hljs-addition">+async function resolvePlugins(config, userPlugins) {</span>
  return [
    preAliasPlugin(config),
    resolvePlugin(config),
<span class="hljs-addition">+   ...userPlugins,</span>
    importAnalysisPlugin(config)
  ]
}
exports.resolvePlugins = resolvePlugins;
</code></pre><p>lib\utils.js</p><pre><code class="lang-diff">function normalizePath(id) {
  return id.replace(/\\/g, '/')
}
exports.normalizePath = normalizePath;
<span class="hljs-addition">+const knownJsSrcRE = /\.(js|vue)/</span>
const isJSRequest = (url) => {
  if (knownJsSrcRE.test(url)) {
    return true
  }
  return false
}
exports.isJSRequest = isJSRequest;
</code></pre><p>lib\config.js</p><pre><code class="lang-diff">const path = require('path');
const { normalizePath } = require('./utils');
const { resolvePlugins } = require('./plugins');
<span class="hljs-addition">+const fs = require('fs-extra');</span>
async function resolveConfig() {
  //当前的根目录 window \\  linux /
  const root = normalizePath(process.cwd());
  const cacheDir = normalizePath(path.resolve(`node_modules/.vite7`))
  let config = {
    root,
    cacheDir
  }
<span class="hljs-addition">+ const jsconfigFile = path.resolve(root, 'vite.config.js')</span>
<span class="hljs-addition">+ const exists = await fs.pathExists(jsconfigFile)</span>
<span class="hljs-addition">+ if (exists) {</span>
<span class="hljs-addition">+   const userConfig = require(jsconfigFile);</span>
<span class="hljs-addition">+   config = { ...config, ...userConfig };</span>
<span class="hljs-addition">+ }</span>
<span class="hljs-addition">+ const userPlugins = config.plugins || [];</span>
<span class="hljs-addition">+ const plugins = await resolvePlugins(config, userPlugins);</span>
  config.plugins = plugins;
  return config;
}
module.exports = resolveConfig;
</code></pre><p>index.html</p><pre><code class="lang-html">
<span class="hljs-tag"><<span class="hljs-name">html</span> <span class="hljs-attr">lang</span>=<span class="hljs-string">"en"</span>></span>

<span class="hljs-tag"><<span class="hljs-name">head</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">meta</span> <span class="hljs-attr">charset</span>=<span class="hljs-string">"UTF-8"</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">meta</span> <span class="hljs-attr">http-equiv</span>=<span class="hljs-string">"X-UA-Compatible"</span> <span
class="hljs-attr">content</span>=<span class="hljs-string">"IE=edge"</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">meta</span> <span class="hljs-attr">name</span>=<span class="hljs-string">"viewport"</span> <span class="hljs-
attr">content</span>=<span class="hljs-string">"width=device-width, initial-scale=1.0"</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">title</span>></span>Document<span class="hljs-tag"></<span class="hljs-name">title</span>></span>
<span class="hljs-tag"></<span class="hljs-name">head</span>></span>

<span class="hljs-tag"><<span class="hljs-name">body</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">div</span> <span class="hljs-attr">id</span>=<span class="hljs-string">"app"</span>></span><span class="hljs-
tag"></<span class="hljs-name">div</span>></span>
  <span class="hljs-tag"><<span class="hljs-name">script</span> <span class="hljs-attr">src</span>=<span class="hljs-string">"/src/main.js"</span> <span
class="hljs-attr">type</span>=<span class="hljs-string">"module"</span>></span><span class="hljs-tag"></<span class="hljs-name">script</span>></span>
<span class="hljs-tag"></<span class="hljs-name">body</span>></span>

<span class="hljs-tag"></<span class="hljs-name">html</span>></span>
</code></pre><p>src\main.js</p><pre><code class="lang-js"><span class="hljs-keyword">import</span> { createApp } <span class="hljs-keyword">from</span> <span
class="hljs-string">'vue'</span>;
<span class="hljs-keyword">import</span> App <span class="hljs-keyword">from</span> <span class="hljs-string">'/src/App.vue'</span>;
createApp(App).mount(<span class="hljs-string">"#app"</span>);
</code></pre><p>src\App.vue</p><pre><code class="lang-js"><template>
  <h1>App</h1>
</template>
<script>
export default {
  name: 'App'
```

```
}
```

plugins\vue.js

```
const { parse, compileScript, rewriteDefault, compileTemplate } = require('vue/compiler-sfc');
const fs = require('fs');
const descriptorCache = new Map();
function vue() {
  return {
    name: 'vue',
    async transform(code, id) {
      const { filename } = parseVueRequest(id);
      if (filename.endsWith('.vue')) {
        let result = await transformMain(code, filename);
        return result;
      }
      return null;
    }
  }
}

async function getDescriptor(filename) {
  let descriptor = descriptorCache.get(filename);
  if (descriptor) return descriptor;
  const content = await fs.promises.readFile(filename, 'utf8');
  const result = parse(content, { filename });
  descriptor = result.descriptor;
  descriptorCache.set(filename, descriptor);
  return descriptor;
}
async function transformMain(source, filename) {
  const descriptor = await getDescriptor(filename);
  const scriptCode = genScriptCode(descriptor, filename)
  const templateCode = genTemplateCode(descriptor, filename);
  let resolvedCode = [
    templateCode,
    scriptCode,
    `_sfc_main['render'] = render`,
    `export default _sfc_main`
  ].join('\n');
  return { code: resolvedCode }
}

function genScriptCode(descriptor, id) {
  let scriptCode = ''
  let script = compileScript(descriptor, { id });
  if (!script.lang) {
    scriptCode = rewriteDefault(
      script.content,
      '_sfc_main',
    )
  }
  return scriptCode;
}
function genTemplateCode(descriptor, id) {
  let content = descriptor.template.content;
  const result = compileTemplate({ source: content, id });
  return result.code;
}
function parseVueRequest(id) {
  const [filename, querystring = ''] = id.split('?');
  let query = new URLSearchParams(querystring);
  return {
    filename, query
  };
}
module.exports = vue;
```

9.支持style

lib\config.js

```
const path = require('path');
const { normalizePath } = require('./utils');
const { resolvePlugins } = require('./plugins');
const fs = require('fs-extra');
async function resolveConfig() {
  //当前的根目录 window \\  linux /
  const root = normalizePath(process.cwd());
  const cacheDir = normalizePath(path.resolve(`node_modules/.vite7`))
  let config = {
    root,
    cacheDir
  }
  const jsconfigFile = path.resolve(root, 'vite.config.js')
  const exists = await fs.pathExists(jsconfigFile)
  if (exists) {
    const userConfig = require(jsconfigFile);
    config = { ...config, ...userConfig };
  }
  const userPlugins = config.plugins || [];
+ for (const plugin of userPlugins) {
+   if (plugin.config) {
+     const res = await plugin.config(config)
+     if (res) {
+       config = { ...config, ...res }
+     }
+   }
+ }
  const plugins = await resolvePlugins(config, userPlugins);
  config.plugins = plugins;
  return config;
}
module.exports = resolveConfig;
```

src\App.vue

```

  App

export default {
  name: 'App'
}

+</span>
<span class="hljs-addition">+h1 {</span>
<span class="hljs-addition">+  color: red;</span>
<span class="hljs-addition">+}</span>
<span class="hljs-addition">+
```

plugins\vue.js

```
+const { parse, compileScript, rewriteDefault, compileTemplate, compileStyleAsync } = require('vue/compiler-sfc');
const fs = require('fs');
const path = require('path');
+const hash = require('hash-sum');
+const descriptorCache = new Map();
function vue() {
  let root;
  return {
    name: 'vue',
+   config(config) {
+     root = config.root;
+     console.log(root, 'root');
+   },
+   async load(id) {
+     const { filename, query } = parseVueRequest(id);
+     if (query.has('vue')) {
+       const descriptor = await getDescriptor(filename, root);
+       if (query.get('type') === 'style') {
+         let block = descriptor.styles[Number(query.get('index'))];
+         if (block) {
+           return { code: block.content };
+         }
+       }
+     }
+   },
    async transform(code, id) {
+     const { filename, query } = parseVueRequest(id);
+     if (filename.endsWith('.vue')) {
+       if (query.get('type') === 'style') {
+         const descriptor = await getDescriptor(filename, root);
+         let result = await transformStyle(code, descriptor, query.get('index'));
+         return result;
+       } else {
+         let result = await transformMain(code, filename);
+         return result;
+       }
+     }
      return null;
    }
  }
}
+async function transformStyle(code, descriptor, index) {
+  const block = descriptor.styles[index];
+  //如果是CSS，其实翻译之后和翻译之前内容是一样的
+  const result = await compileStyleAsync({
+    filename: descriptor.filename,
+    source: code,
+    id: `data-v-${descriptor.id}`,//必须传递，不然报错
+    scoped: block.scoped
+  });
+  let styleCode = result.code;
+  const injectCode =
+    `\nvar  style = document.createElement('style');` +
+    `\nstyle.innerHTML = ${JSON.stringify(styleCode)};` +
+    `\ndocument.head.appendChild(style);`
+  return {
+    code: injectCode
+  };
+}
+async function getDescriptor(filename, root) {
+  let descriptor = descriptorCache.get(filename);
+  if (descriptor) return descriptor;
+  const content = await fs.promises.readFile(filename, 'utf8');
+  const result = parse(content, { filename });
+  descriptor = result.descriptor;
+  descriptor.id = hash(path.relative(root, filename));
+  descriptorCache.set(filename, descriptor);
+  return descriptor;
+}
async function transformMain(source, filename) {
  const descriptor = await getDescriptor(filename, root);
  const scriptCode = genScriptCode(descriptor, filename)
  const templateCode = genTemplateCode(descriptor, filename);
+ const stylesCode = genStyleCode(descriptor, filename);
  let resolvedCode = [
+   stylesCode,
    templateCode,
    scriptCode,
    `_sfc_main['render'] = render`,
    `export default _sfc_main`
  ].join('\n');
  return { code: resolvedCode }
}
+function genStyleCode(descriptor, filename) {
+  let styleCode = '';
+  if (descriptor.styles.length) {
+    descriptor.styles.forEach((style, index) => {
+      const query = `?vue&type=style&index=${index}&lang=css`;
+      const styleRequest = (filename + query).replace(/\\/g, '/');
+      styleCode += `\nimport ${JSON.stringify(styleRequest)}`;
+    });
+    return styleCode;
+  }
+}
function genScriptCode(descriptor, id) {
  let scriptCode = ''
  let script = compileScript(descriptor, { id });
  if (!script.lang) {
    scriptCode = rewriteDefault(
      script.content,
      '_sfc_main',
    )
  }
  return scriptCode;
}
function genTemplateCode(descriptor, id) {
  let content = descriptor.template.content;
  const result = compileTemplate({ source: content, id });
  return result.code;
}
+function parseVueRequest(id) {
+  const [filename, querystring = ''] = id.split('?');
+  let query = new URLSearchParams(querystring);
+  return {
+    filename, query
+  };
+}
module.exports = vue;
```

10.支持环境变量

lib\plugins\index.js

```
const importAnalysisPlugin = require('./importAnalysis');
const preAliasPlugin = require('./preAlias');
const resolvePlugin = require('./resolve');
const definePlugin = require('./define');
async function resolvePlugins(config, userPlugins) {
  return [
    preAliasPlugin(config),
    resolvePlugin(config),
    ...userPlugins,
    definePlugin(config),
    importAnalysisPlugin(config)
  ]
}
exports.resolvePlugins = resolvePlugins;
```

lib\plugins\define.js

```
const MagicString = require('magic-string');
function definePlugin(config) {
  return {
    name: 'vite:define',
    transform(code) {
      const replacements = config.define || {};
      const replacementsKeys = Object.keys(replacements)
      const pattern = new RegExp('(' + replacementsKeys.map(str => str).join('|') + ')', 'g');
      const s = new MagicString(code)
      let hasReplaced = false
      let match
      while ((match = pattern.exec(code))) {
        hasReplaced = true
        const start = match.index
        const end = start + match[0].length
        const replacement = '' + replacements[match[1]]
        s.overwrite(start, end, replacement)
      }
      if (!hasReplaced) {
        return null
      }
      return { code: s.toString() }
    }
  }
}
module.exports = definePlugin;
```

plugins\vue.js

```
const { parse, compileScript, rewriteDefault, compileTemplate, compileStyleAsync } = require('vue/compiler-sfc');
const fs = require('fs');
const path = require('path');
const hash = require('hash-sum');
const descriptorCache = new Map();
function vue() {
  let root;
  return {
    name: 'vue',
    config(config) {
      root = config.root;
+     return {
+       define: {
+         __VUE_OPTIONS_API__: true,
+         __VUE_PROD_DEVTOOLS__: false
+       }
+     }
    },
    async load(id) {
      const { filename, query } = parseVueRequest(id);
      if (query.has('vue')) {
        const descriptor = await getDescriptor(filename, root);
        if (query.get('type')
          let block = descriptor.styles[Number(query.get('index'))];
          if (block) {
            return { code: block.content };
          }
        }
      }
    },
    async transform(code, id) {
      const { filename, query } = parseVueRequest(id);
      if (filename.endsWith('.vue')) {
        if (query.get('type')
          const descriptor = await getDescriptor(filename, root);
          let result = await transformStyle(code, descriptor, query.get('index'));
          return result;
        } else {
          let result = await transformMain(code, filename);
          return result;
        }
      }
      return null;
    }
  }
}
async function transformStyle(code, descriptor, index) {
  const block = descriptor.styles[index];
  //如果是CSS，其实翻译之后和翻译之前内容是一样的，最终返回的JS靠packages\vite\src\node\plugins\css.ts
  const result = await compileStyleAsync({
    filename: descriptor.filename,
    source: code,
    id: `data-v-${descriptor.id}`,//必须传递，不然报错
    scoped: block.scoped
  });
  let styleCode = result.code;
  const injectCode =
    `\nvar  style = document.createElement('style');` +
    `\nstyle.innerHTML = ${JSON.stringify(styleCode)};` +
    `\ndocument.head.appendChild(style);`
  return {
    code: injectCode
  };
}
async function getDescriptor(filename, root) {
  let descriptor = descriptorCache.get(filename);
  if (descriptor) return descriptor;
  const content = await fs.promises.readFile(filename, 'utf8');
  const result = parse(content, { filename });
  descriptor = result.descriptor;
  descriptor.id = hash(path.relative(root, filename));
  descriptorCache.set(filename, descriptor);
  return descriptor;
}
async function transformMain(source, filename) {
  const { descriptor } = parse(source, { filename });
  const scriptCode = genScriptCode(descriptor, filename)
  const templateCode = genTemplateCode(descriptor, filename);
  const stylesCode = genStyleCode(descriptor, filename);
  let resolvedCode = [
    stylesCode,
    templateCode,
    scriptCode,
    `_sfc_main['render'] = render`,
    `export default _sfc_main`
  ].join('\n');
  return { code: resolvedCode }
}
function genStyleCode(descriptor, filename) {
  let styleCode = '';
  if (descriptor.styles.length) {
    descriptor.styles.forEach((style, index) => {
      const query = `?vue&type=style&index=${index}&lang=css`;
      const styleRequest = (filename + query).replace(/\\/g, '/');
      styleCode += `\nimport ${JSON.stringify(styleRequest)}`;
    });
    return styleCode;
  }
}
function genScriptCode(descriptor, id) {
  let scriptCode = ''
  let script = compileScript(descriptor, { id });
  if (!script.lang) {
    scriptCode = rewriteDefault(
      script.content,
      '_sfc_main',
    )
  }
  return scriptCode;
}
function genTemplateCode(descriptor, id) {
  let content = descriptor.template.content;
  const result = compileTemplate({ source: content, id });
  return result.code;
}
function parseVueRequest(id) {
  const [filename, querystring = ''] = id.split('?');
  let query = new URLSearchParams(querystring);
  return {
    filename, query
  };
}
module.exports = vue;
```

11.支持HMR

lib\server\index.js

```
const connect = require('connect');
const http = require('http');
const serveStaticMiddleware = require('./middlewares/static');
const resolveConfig = require('../config');
const { createOptimizeDepsRun } = require('../optimizer');
const transformMiddleware = require('./middlewares/transform');
const { createPluginContainer } = require('./pluginContainer');
+const { handleHMRUpdate } = require('./hmr');
+const { createWebSocketServer } = require('./ws');
+const { normalizePath } = require('../utils');
+const chokidar = require('chokidar');
+const { ModuleGraph } = require('./moduleGraph')
+const path = require('path');
async function createServer() {
  const config = await resolveConfig();
  const middlewares = connect();
+ const httpServer = require('http').createServer(middlewares)
+ const ws = createWebSocketServer(httpServer, config)
+ const watcher = chokidar.watch(path.resolve(config.root), {
+   ignored: [
+     '**/node_modules/**',
+     '**/.git/**'
+   ]
+ });
+ const moduleGraph = new ModuleGraph((url) =>
+   pluginContainer.resolveId(url)
+ )
+ const pluginContainer = await createPluginContainer(config)
  const server = {
+   config,
+   ws,
+   watcher,
+   moduleGraph,
+   httpServer,
    pluginContainer,
    async listen(port) {
      await runOptimize(config, server)
+     httpServer.listen(port, async () => {
        console.log(`server running at http://localhost:${port}`);
      });
    }
  }
+ watcher.on('change', async (file) => {
+   file = normalizePath(file)
+   await handleHMRUpdate(file, server)
+ })
  for (const plugin of config.plugins) {
    if (plugin.configureServer) {
      await plugin.configureServer(server)
    }
  }
  middlewares.use(transformMiddleware(server))
  middlewares.use(serveStaticMiddleware(config));
  return server;
}
async function runOptimize(config, server) {
  const optimizeDeps = await createOptimizeDepsRun(config);
  server._optimizeDepsMetadata = optimizeDeps.metadata
}
exports.createServer = createServer;
```

lib\server\ws.js

```
const { WebSocketServer } = require('ws');
const HMR_HEADER = 'vite-hmr'
function createWebSocketServer(httpServer) {
  const wss = new WebSocketServer({ noServer: true });
  httpServer.on('upgrade', (req, socket, head) => {
    if (req.headers['sec-websocket-protocol'] === HMR_HEADER) {
      wss.handleUpgrade(req, socket, head, (ws) => {
        wss.emit('connection', ws, req)
      })
    }
  })
  wss.on('connection', (socket) => {
    socket.send(JSON.stringify({ type: 'connected' }))
  })
  return {
    on: wss.on.bind(wss),
    off: wss.off.bind(wss),
    send(payload) {
      const stringified = JSON.stringify(payload)
      wss.clients.forEach((client) => {
        if (client.readyState === 1) {
          client.send(stringified)
        }
      })
    }
  }
}
exports.createWebSocketServer = createWebSocketServer;
```

lib\server\hmr.js

```
const path = require('path');
const LexerState = {
  inCall: 0,
  inSingleQuoteString: 1,
  inTemplateString: 2
}
function getShortName(file, root) {
  return file.startsWith(root + '/') ? path.posix.relative(root, file) : file
}
async function handleHMRUpdate(file, server) {
  const { config, moduleGraph } = server
  const shortFile = getShortName(file, config.root)
  const modules = moduleGraph.getModulesByFile(file) || []
  updateModules(shortFile, modules, server)
}

function updateModules(file, modules, { ws }) {
  const updates = []
  for (const mod of modules) {
    const boundaries = new Set()
    propagateUpdate(mod, boundaries)
    updates.push(
      ...[...boundaries].map(({ boundary, acceptedVia }) => ({
        type: `${boundary.type}-update`,
        path: boundary.url,
        acceptedPath: acceptedVia.url
      }))
    )
  }
  ws.send({
    type: 'update',
    updates
  })
}
function propagateUpdate(node, boundaries) {
  if (!node.importers.size) {
    return true
  }
  for (const importer of node.importers) {
    if (importer.acceptedHmrDeps.has(node)) {
      boundaries.add({
        boundary: importer,
        acceptedVia: node
      })
      continue
    }
  }
  return false;
}
function lexAcceptedHmrDeps(code, start, urls) {
  let state = LexerState.inCall
  let prevState = LexerState.inCall
  let currentDep = ''
  function addDep(index) {
    urls.add({
      url: currentDep,
      start: index - currentDep.length - 1,
      end: index + 1
    })
    currentDep = ''
  }
  for (let i = start; i < code.length; i++) {
    const char = code.charAt(i)
    switch (state) {
      case LexerState.inCall:
        if (char === `'`) {
          prevState = state
          state = LexerState.inSingleQuoteString
        }
        break
      case LexerState.inSingleQuoteString:
        if (char === `'`) {
          addDep(i)
          return false
        } else {
          currentDep += char
        }
        break
      default:
        break;
    }
  }
  return false
}
exports.handleHMRUpdate = handleHMRUpdate;
exports.updateModules = updateModules;
exports.lexAcceptedHmrDeps = lexAcceptedHmrDeps;
```

lib\server\transformRequest.js

```
const fs = require('fs-extra');
async function transformRequest(url, server) {
  const { pluginContainer } = server
  const { id } = await pluginContainer.resolveId(url);
  const loadResult = await pluginContainer.load(id)
  let code;
  if (loadResult) {
    code = loadResult.code;;
  } else {
    code = await fs.readFile(id, 'utf-8')
  }
+ await server.moduleGraph.ensureEntryFromUrl(url)
  const transformResult = await pluginContainer.transform(code, id)
  return transformResult;
}
module.exports = transformRequest;
```

lib\server\moduleGraph.js

```
const path = require('path');
class ModuleNode {
  importers = new Set()
  acceptedHmrDeps = new Set()
  constructor(url) {
    this.url = url
    this.type = 'js'
  }
}
class ModuleGraph {
  constructor(resolveId) {
    this.resolveId = resolveId;
  }

  fileToModulesMap = new Map()
  urlToModuleMap = new Map()
  idToModuleMap = new Map()

  getModulesByFile(file) {
    return this.fileToModulesMap.get(file)
  }
  getModuleById(id) {
    return this.idToModuleMap.get(id)
  }
  async ensureEntryFromUrl(rawUrl) {
    const [url, resolvedId] = await this.resolveUrl(rawUrl)
    let mod = this.urlToModuleMap.get(url)
    if (!mod) {
      mod = new ModuleNode(url)
      this.urlToModuleMap.set(url, mod)
      this.idToModuleMap.set(resolvedId, mod)
      const file = (mod.file = resolvedId)
      let fileMappedModules = this.fileToModulesMap.get(file)
      if (!fileMappedModules) {
        fileMappedModules = new Set()
        this.fileToModulesMap.set(file, fileMappedModules)
      }
      fileMappedModules.add(mod)
    }
    return mod;
  }
  async resolveUrl(url) {
    const resolved = await this.resolveId(url)
    const resolvedId = resolved.id || url
    return [url, resolvedId]
  }
  async updateModuleInfo(mod, importedModules, acceptedModules) {
    for (const imported of importedModules) {
      const dep = await this.ensureEntryFromUrl(imported)
      dep.importers.add(mod)
    }
    const deps = (mod.acceptedHmrDeps = new Set())
    for (const accepted of acceptedModules) {
      const dep = await this.ensureEntryFromUrl(accepted)
      deps.add(dep)
    }
  }
}
exports.ModuleGraph = ModuleGraph;
```

lib\plugins\importAnalysis.js

```
const { init, parse } = require('es-module-lexer')
const MagicString = require('magic-string');
+const { lexAcceptedHmrDeps } = require('../server/hmr');
+const path = require('path');
function importAnalysisPlugin(config) {
  const { root } = config
+ let server
  return {
    name: 'vite:import-analysis',
+   configureServer(_server) {
+     server = _server
+   },
    async transform(source, importer) {
      await init
      let imports = parse(source)[0]
      if (!imports.length) {
        return source
      }
+     const { moduleGraph } = server
+     const importerModule = moduleGraph.getModuleById(importer)
+     const importedUrls = new Set()
+     const acceptedUrls = new Set()
      let ms = new MagicString(source);
      const normalizeUrl = async (url) => {
        const resolved = await this.resolve(url, importer)
        if (resolved.id.startsWith(root + '/')) {
          url = resolved.id.slice(root.length)
        }
+       await moduleGraph.ensureEntryFromUrl(url)
        return url;
      }
      for (let index = 0; index < imports.length; index++) {
        const { s: start, e: end, n: specifier } = imports[index]
        const rawUrl = source.slice(start, end)
+       if (rawUrl === 'import.meta') {
+         const prop = source.slice(end, end + 4)
+         if (prop === '.hot') {
+           if (source.slice(end + 4, end + 11) === '.accept') {
+             lexAcceptedHmrDeps(source, source.indexOf('(', end + 11) + 1, acceptedUrls)
+           }
+         }
+       }
        if (specifier) {
          const normalizedUrl = await normalizeUrl(specifier)
          if (normalizedUrl !== specifier) {
            ms.overwrite(start, end, normalizedUrl)
          }
+         importedUrls.add(normalizedUrl)
        }
      }
+     const normalizedAcceptedUrls = new Set()
+     const toAbsoluteUrl = (url) =>
+       path.posix.resolve(path.posix.dirname(importerModule.url), url)
+     for (const { url, start, end } of acceptedUrls) {
+       const [normalized] = await moduleGraph.resolveUrl(toAbsoluteUrl(url),)
+       normalizedAcceptedUrls.add(normalized)
+       ms.overwrite(start, end, JSON.stringify(normalized))
+     }
+     await moduleGraph.updateModuleInfo(
+       importerModule,
+       importedUrls,
+       normalizedAcceptedUrls
+     )
      return ms.toString()
    }
  }
}
module.exports = importAnalysisPlugin;
```

index.html

```

<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Documenttitle>
head>

<body>
  <div id="app">div>
  <script src="/src/main.js" type="module">script>
  [illegible]
[illegible]

[illegible]
```

src\main.js

```
import { render } from './render.js';
render();
window.hotModulesMap = new Map()
var ownerPath = "/src/main.js";
import.meta.hot = {
  accept(deps, callback) {
    acceptDeps(deps, callback)
  }
}
function acceptDeps(deps, callback) {
  const mod = hotModulesMap.get(ownerPath) || {
    id: ownerPath,
    callbacks: []
  }
  mod.callbacks.push({
    deps,
    fn: callback
  })
  hotModulesMap.set(ownerPath, mod)
}
if (import.meta.hot) {
  import.meta.hot.accept(['./render.js'], ([renderMod]) => {
    renderMod.render();
  });
}
```

src\render.js

```
export function render() {
  app.innerHTML = 'title1';
}
```

src\client.js

```
console.log('[vite] connecting...')
var socket = new WebSocket(`ws://${window.location.host}`, 'vite-hmr')
socket.addEventListener('message', async ({ data }) => {
  handleMessage(JSON.parse(data))
})
async function handleMessage(payload) {
  switch (payload.type) {
    case 'connected':
      console.log(`[vite] connected.`)
      break;
    case 'update':
      payload.updates.forEach((update) => {
        if (update.type === 'js-update') {
          fetchUpdate(update)
        }
      });
      break;
    case 'full-reload':
      location.reload()
    default:
      break;
  }
}

async function fetchUpdate({ path, acceptedPath }) {
  const mod = window.hotModulesMap.get(path)
  if (!mod) {
    return
  }
  const moduleMap = new Map()
  const modulesToUpdate = new Set()
  for (const { deps } of mod.callbacks) {
    deps.forEach((dep) => {
      if (acceptedPath === dep) {
        modulesToUpdate.add(dep)
      }
    })
  }
  await Promise.all(
    Array.from(modulesToUpdate).map(async (dep) => {
      const newMod = await import(dep + '?ts=' + Date.now())
      moduleMap.set(dep, newMod)
    })
  )
  for (const { deps, fn } of mod.callbacks) {
    fn(deps.map((dep) => moduleMap.get(dep)))
  }
  const loggedPath = `${acceptedPath} via ${path}`
  console.log(`[vite] hot updated: ${loggedPath}`)
}
```