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=210 sentences=997, words=7287

1.核心知识

1.1 安装依赖

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

1.2 Connect

- 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 连接在一起
- 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);
```

1.3 serve-static

- serve-static (https://www.npmjs.com/package/serve-static)是一个静态文件中中间件

```
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);
```

1.4 es-module-lexer

- es-module-lexer (https://www.npmjs.com/package/es-module-lexer)是一个 JS 模块语法解析器

```
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);
})();
```

1.5 resolve

- resolve (https://www.npmjs.com/package/resolve)实现了 node 的 require.resolve() 算法

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

1.6 fast-glob

- fast-glob (https://www.npmjs.com/package/fast-glob)该包提供了一些方法，用于遍历文件系统，并根据 Unix Bash shell使用的规则返回与指定模式的定义集匹配的路径名

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

1.7 magic-string

- magic-string (https://www.npmjs.com/package/magic-string)是一个用来操作字符串的库

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

2.实现命令行

2.1 package.json

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

2.2 vite3.js

bin\vite3.js

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

2.3 cli.js

lib\cli.js

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

3.实现 http 服务器

3.1 cli.js

lib\cli.js

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

3.2 server\index.js

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

4.1 server\index.js

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;
```

4.2 static.js

lib\server\middlewares\static.js

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

4.3 config.js

lib\config.js

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

4.4 utils.js

lib\utils.js

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

4.5 index.html

viteuse\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 type="module" src="/src/main.js">script>
  body>
html>
```

4.6 main.js

viteuse\main.js

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

5.分析第三方依赖

5.1 main.js

src\main.js

```
+import { createApp } from 'vue'
+console.log(createApp);
```

5.2 server\index.js

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;
```

5.3 optimizer\index.js

lib\optimizer\index.js

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

5.4 scan.js

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;
```

5.5 esbuildScanPlugin.js

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>
```

```
(config, depImports) ){
  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) {
      build.onResolve({ filter: htmlTypesRE }, async ({ path, importer }) => {
        const resolved = await resolve(path, importer);
        if (resolved) {
          return {
            path: resolved.id || resolved,
            namespace: "html",
          };
        }
      });
      build.onResolve({ filter: /.*/ }, async ({ path, importer }) => {
        const resolved = await resolve(path, importer);
        if (resolved) {
          const id = resolved.id || resolved;
          const included = id.includes("node_modules");
          if (included) {
            depImports[path] = normalizePath(id);
            return {
              path: id,
              external: true,
            };
          }
          return {
            path: id,
          };
        }
        return { path };
      });
      build.onLoad(
        { filter: htmlTypesRE, namespace: "html" },
        async ({ path }) => {
          let html = fs.readFileSync(path, "utf-8");
          let [, scriptSrc] = html.match(scriptModuleRE);
          let js = `import ${JSON.stringify(scriptSrc)}`;\n`;
          return {
            loader: "js",
            contents: js,
          };
        }
      );
      build.onLoad({ filter: JS_TYPES_RE }, ({ path: id }) => {
        const ext = path.extname(id).slice(1);
        const contents = fs.readFileSync(id, "utf-8");
        return {
          loader: ext,
          contents,
        };
      });
    },
  };
}
module.exports = esbuildScanPlugin;
```

</code></pre><h3 id="t285.6 pluginContainer.js">5.6 pluginContainer.js #</h3> lib\server\pluginContainer.js <pre><code class="lang-js">

```
const { normalizePath } = require("../utils");
const path = require("path");
async function createPluginContainer({ plugins, root }) {
  class PluginContext {}
  const container = {
    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) };
    },
  };
  return container;
}
exports.createPluginContainer = createPluginContainer;
```

</code></pre><h3 id="t295.7 resolve.js">5.7 resolve.js #</h3> lib\plugins\resolve.js <pre><code class="lang-js">

```
const fs = require("fs");
const path = require("path");
const resolve = require("resolve");
function resolvePlugin(config) {
  return {
    name: "vite:resolve",
    resolveId(id, importer) {

      if (id.startsWith("/")) {
        return { id: path.resolve(config.root, id.slice(1)) };
      }

      if (path.isAbsolute(id)) {
        return { 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;
```

6.1 server\index.js

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

6.2 config.js

lib\config.js

```
+const path = require('path');
const { normalizePath } = require('./utils');
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
  }
  return config;
}
module.exports = resolveConfig;
```

6.3 optimizer\index.js

lib\optimizer\index.js

```
const scanImports = require('./scan');
+const fs = require('fs-extra');
+const path = require('path');
+const { build } = require('esbuild');
+const { normalizePath } = require('../utils');
async function createOptimizeDepsRun(config) {
  const deps = await scanImports(config);
+ const { cacheDir } = config;
+ const depsCacheDir = path.resolve(cacheDir, 'deps')
+ const metadataPath = path.join(depsCacheDir, '_metadata.json');
+ const metadata = {
+   optimized: {}
+ }
+ for (const id in deps) {
+   const entry = deps[id]
+   metadata.optimized[id] = {
+     file: normalizePath(path.resolve(depsCacheDir, id + '.js')),
+     src: entry
+   }
+   await build({
+     absWorkingDir: process.cwd(),
+     entryPoints: [deps[id]],
+     outfile: path.resolve(depsCacheDir, id + '.js'),
+     bundle: true,
+     write: true,
+     format: 'esm'
+   })
+ }
+ await fs.ensureDir(depsCacheDir);
+ await fs.writeFile(metadataPath, JSON.stringify(metadata, (key, value) => {
+   if (key === 'file' || key === 'src') {
+     //optimized里存的是绝对路径，此处写入硬盘的是相对于缓存目录的相对路径
+     console.log(depsCacheDir, value);
+     return normalizePath(path.relative(depsCacheDir, value));
+   }
+   return value
+ }, 2));
```

```
<span class="hljs-addition">+ return { metadata };</span>
}
exports.createOptimizeDepsRun = createOptimizeDepsRun;
</code></pre><h2 id="t347.修改导入路径">7.修改导入路径 <a href="#t347.修改导入路径">#</a></h2><ul><li><p>修改返回的 <code>main.js</code> 中的 <code>vue</code> 的路径</p>
<ul><li><code>import { createApp } from 'vue'</code> 变为</li><li><code>import { createApp } from '/node_modules/.vite/deps/vue.js'</code></li></ul></li><li><p>
请求<code>/src/main.js</code>,此请求先由<code>transformMiddleware</code>中间件处理，通过<code>isJSRequest</code>判断是 js 请求,走<code>transformRequest</code></p></li>
<li>在<code>transformRequest</code>里执行<code>pluginContainer.resolveId(url)</code>方法,方法内会由<code>resolvePlugin</code>返回<code>/src/main.js</code>的绝对路径
</li><li>再调用<code>pluginContainer.load(id)</code>方法返回 JS 文件内容,此处返回<code>null</code></li><li>再调用<code>pluginContainer.transform(code, id)</code>方法,
执行<code>importAnalysisPlugin</code>里的<code>transform</code>方法,里面会分析依赖的模块，获取依赖的<code>vue</code>,重新执行<code>PluginContext.resolve</code>,执
行<code>preAliasPlugin</code>,获取<code>vue</code>相对路径<code>/node_modules/.vite/deps/vue.js</code>,把<code>vue</code>变为
<code>/node_modules/.vite/deps/vue.js</code></li></ul><p><img src="https://static.zhufengpeixun.com/xiu_gai_dao_ru_lu_jing_1663307154476.jpg"></p><h3 id="t357.1
server\index.js">7.1 server\index.js <a href="#t357.1 server\index.js">#</a></h3><ul><li>使用转换请求的<code>transformMiddleware</code>中间件</li><li>执
行<code>transformRequest</code><ul><li><code>pluginContainer.resolveId</code></li></ul></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><h3 id="t367.2 transform.js">7.2 transform.js <a href="#t367.2 transform.js">#</a></h3><ul><li>判断如果请求的是 JS 文件的请就进行 JS 内容转换</li></ul>
<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(req.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><h3 id="t377.3 utils.js">7.3 utils.js <a href="#t377.3 utils.js">#</a></h3><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><h3 id="t387.4 transformRequest.js">7.4 transformRequest.js <a href="#t387.4 transformRequest.js">#</a></h3><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;
```

```
}
module.exports = transformRequest;
</code></pre><h3 id="t397.5 pluginContainer.js">7.5 pluginContainer.js <a href="#t397.5 pluginContainer.js">#</a></h3><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 {
+   async resolve(id, importer = path.join(root, "index.html")) {
+     return await container.resolveId(id, importer);
+   }
  }
  //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) }
    },
+   async load(id) {
+     const ctx = new PluginContext()
+     for (const plugin of plugins) {
+       if (!plugin.load) continue
+       const result = await plugin.load.call(ctx, id)
+       if (result !== null) {
+         return result
+       }
+     }
+     return null
+   },
+   async transform(code, id) {
+     for (const plugin of plugins) {
+       if (!plugin.transform) continue
+       const ctx = new PluginContext()
+       const result = await plugin.transform.call(ctx, code, id)
+       if (!result) continue
+       code = result.code || result;
+     }
+     return { code }
+   }
  }
  return container;
}
exports.createPluginContainer = createPluginContainer;
</code></pre><h3 id="t407.6 send.js">7.6 send.js <a href="#t407.6 send.js">#</a></h3><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><h3 id="t417.7 plugins\index.js">7.7 plugins\index.js <a href="#t417.7 plugins\index.js">#</a></h3><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><h3 id="t427.8 importAnalysis.js">7.8 importAnalysis.js <a href="#t427.8 importAnalysis.js">#</a></h3><ul><li>导入文件分析
lib\plugins\importAnalysis.js</li></ul><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><h3 id="t437.9 preAlias.js">7.9 preAlias.js <a href="#t437.9 preAlias.js">#</a></h3><ul><li>看看是否是经过预构建的路径，如果是直接取预构建后的路径</li></ul>
<p>lib\plugins\preAlias.js</p><pre><code class="lang-js"><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><h3 id="t447.10 config.js">7.10 config.js <a href="#t447.10 config.js">#</a></h3><p>lib\config.js</p><pre><code class="lang-diff">const path =
require('path');
const { normalizePath } = require('./utils');
<span class="hljs-addition">+const { resolvePlugins } = require('./plugins');</span>
async function resolveConfig() {
  const root = normalizePath(process.cwd());
  const cacheDir = normalizePath(path.resolve(`node_modules/.vite`))
  let config = {
    root,
    cacheDir
  }
<span class="hljs-addition">+ const plugins = await resolvePlugins(config);</span>
<span class="hljs-addition">+ config.plugins = plugins;</span>
  return config;
}
module.exports = resolveConfig;
</code></pre><h2 id="t458.支持 vue 插件">8.支持 vue 插件 <a href="#t458.支持 vue 插件">#</a></h2><h3 id="t468.1 esbuildDepPlugin.js">8.1 esbuildDepPlugin.js <a
href="#t468.1 esbuildDepPlugin.js">#</a></h3><p>lib\optimizer\esbuildDepPlugin.js</p><pre><code class="lang-diff">const path = require('path');
const fs = require('fs-extra');
const htmlTypesRE = /\.html$/;
const scriptModuleRE = /<script type="module" src\="(.+?)">
const | createPluginContainer | = require('../server/pluginContainer');
const resolvePlugin = require('../plugins/resolve');
const jsMap = /\.js$/;
async function esbuildDepPlugin(config, deps) {
  //自己放置全局单例的vite插件集合
  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) {
      //对 [STDIN] No loader is configured for ".vue" files: src/App.vue
      build.onResolve({ filter: /.*/ }, async ({ path, importer }) => {
        //path:C:\Users\zhufeng\Desktop\vite6\index.html importer 空
        const resolved = await resolve(path, importer);
        if (resolved) {
          return {
            path: resolved.id || resolved,
            namespace: 'dep' //为了更细化区分字符的文件类型，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先走开放原
      build.onResolve({ filter: /.*/ }, async ({ path, importer }) => {
        const resolved = await resolve(path, importer);
```

```
        //这里有可能是 {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: jsTS }, 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><h3 id="t-578.2 plugins\index.js">8.2 plugins\index.js <a href="#t-578.2 plugins\index.js">#</a></h3><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><h3 id="t-588.3 utils.js">8.3 utils.js <a href="#t-588.3 utils.js">#</a></h3><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><h3 id="t-598.4 config.js">8.4 config.js <a href="#t-598.4 config.js">#</a></h3><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><h3 id="t-608.5 index.html">8.5 index.html <a href="#t-608.5 index.html">#</a></h3><p>index.html</p><pre><code class="lang-html">
<span class="hljs-tag">&lt;<span class="hljs-name">html</span> <span class="hljs-attr">lang</span>=<span class="hljs-string">"en"</span>&gt;</span>
  <span class="hljs-tag">&lt;<span class="hljs-name">head</span>&gt;</span>
    <span class="hljs-tag">&lt;<span class="hljs-name">meta</span> <span class="hljs-attr">charset</span>=<span class="hljs-string">"UTF-8"</span> /&gt;</span>
    <span class="hljs-tag">&lt;<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> /&gt;</span>
    <span class="hljs-tag">&lt;<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> /&gt;</span>
    <span class="hljs-tag">&lt;<span class="hljs-name">title</span>&gt;</span>Document<span class="hljs-tag">&lt;/<span class="hljs-name">title</span>&gt;</span>
  <span class="hljs-tag">&lt;/<span class="hljs-name">head</span>&gt;</span>

  <span class="hljs-tag">&lt;<span class="hljs-name">body</span>&gt;</span>
    <span class="hljs-tag">&lt;<span class="hljs-name">div</span> <span class="hljs-attr">id</span>=<span class="hljs-string">"app"</span>&gt;</span><span
class="hljs-tag">&lt;/<span class="hljs-name">div</span>&gt;</span>
```

</pre><h3 id="t518.6 main.js">8.6 main.js #</h3> src\main.js <pre><code class="lang-js">import { createApp } from "vue";
import App from "/src/App.vue";
createApp(App).mount("#app");
</code></pre><h3 id="t528.7 App.vue">8.7 App.vue #</h3> src\App.vue <pre><code class="lang-js"><template>
  <h1>App</h1>
</template>
<script>
export default {
  name: 'App'
}

8.8 vue.js

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;
```

8.9 vite.config.js

vite.config.js

```
const vue = require("./plugins/vue");
module.exports = {
  plugins: [vue()],
};
```

9.支持 style

9.1 config.js

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;
```

9.2 App.vue

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">+
```

9.3 vue.js

plugins\vue.js

```
npm install hash-sum --save
```

```
+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;
+   },
+   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,root);
+         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 = ${JSON.stringify(styleCode)};` +
+    `\ndocument.head.appendChild(style);`
+  return {
+    code: injectCode
+  };
+}
+async function getDescriptor(filename, root) {
+  let descriptor = descriptorCache.get(filename);
+  if (descriptor) return descriptor;
+  const content = await fs.promises.readFile(filename, 'utf8');
+  const result = parse(content, { filename });
+  descriptor = result.descriptor;
+  descriptor.id = hash(path.relative(root, filename));
+  descriptorCache.set(filename, descriptor);
+  return descriptor;
+}
async function transformMain(source, filename,root) {
  const descriptor = await getDescriptor(filename, root);
  const scriptCode = genScriptCode(descriptor, filename)
  const templateCode = genTemplateCode(descriptor, filename);
+ const stylesCode = genStyleCode(descriptor, filename);
  let resolvedCode = [
+   stylesCode,
    templateCode,
    scriptCode,
    `_sfc_main['render'] = render`,
    `export default _sfc_main`
  ].join('\n');
  return { code: resolvedCode }
}
+function genStyleCode(descriptor, filename) {
+  let styleCode = '';
+  if (descriptor.styles.length) {
+    descriptor.styles.forEach((style, index) => {
+      const query = `?vue&type=style&index=${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

10.1 plugins\index.js

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;
```

10.2 define.js

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;
```

10.3 plugins\vue.js

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,root);
          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,root) {
  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.HMR

- HMR (https://cn.vitejs.dev/guide/api-hmr.html)

11.1.创建项目

11.1.1 安装

```
pnpm install vite -D
```

11.1.2.package.json

```
{
  "scripts": {
    "dev": "vite"
  }
}
```

11.1.3.src\main.js

src\main.js

```
export function render() {
  app.innerHTML = 'title';
}
render();
if (import.meta.hot) {
  import.meta.hot.accept((newModule) => {
    newModule.render();
  });
}
```

11.1.4.index.html

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>hmrtitle>
head>

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

html>
```

11.2.封装模块

11.2.1 src\render.js

src\render.js

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

12.2.2 src\main.js

src\main.js

```
import { render } from './render';
render();
if (import.meta.hot) {
  import.meta.hot.accept(['./render'], ([renderMod]) => {
    renderMod.render();
  });
}
```

11.3.销毁副作用

11.3.1 render.js

src\render.js

```
+let counter = { number: 0 };
+let timer = setInterval(() => {
+  console.log(counter.number++);
+}, 1000);

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

+if (import.meta.hot) {
+  import.meta.hot.dispose(() => {
+    console.log('dispose render.js');
+    clearInterval(timer);
+  });
+}
```

11.4.保留状态

11.4.1 render.js

src\render.js

```
+let counter = import.meta.hot.data.counter || { number: 0 };
let timer;

export function render() {
  timer = setInterval(() => {
    app.innerHTML = counter.number++;
  }, 1000);
}

if (import.meta.hot) {
//每个模块有一个data属性,保存热更新前的状态
+ import.meta.hot.data.counter = counter;
  import.meta.hot.dispose(() => {
    console.log('dispose render.js');
    clearInterval(timer);
  });
}
```

11.5.拒绝更新

11.5.1 render.js

src\render.js

```
+export let counter = import.meta.hot.data.counter || { number: 0 };
let timer;

export function render() {
  timer = setInterval(() => {
    app.innerHTML = counter.number++;
  }, 1000);
}

if (import.meta.hot) {
  import.meta.hot.data.counter = counter;
  import.meta.hot.dispose(() => {
    console.log('dispose render.js');
    clearInterval(timer);
  });
}
```

11.5.2 src\main.js

src\main.js

```
import { render } from './render';
render();
if (import.meta.hot) {
  import.meta.hot.accept(['./render'], ([renderMod]) => {
+    if (renderMod.counter.number < 10) {
+      renderMod.render();
+    } else {
+      //强制刷新
+      import.meta.hot.invalidate();
    }
  });
+  //import.meta.hot.accept();
+  import.meta.hot.decline();
}
```

12.支持 HMR

12.1 server\index.js

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;
```

12.2 ws.js

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;
```

12.3 hmr.js

lib\server\hmr.js

```
const LexerState = {
  inCall: 0,
  inQuoteString: 1,
};

async function handleHMRUpdate(file, server) {
  const { moduleGraph, ws } = server;

  const module = moduleGraph.getModuleById(file);
  if (module) {
    const updates = [];
    const boundaries = new Set();
    propagateUpdate(module, boundaries);
    updates.push(
      ...[...boundaries].map(({ boundary, acceptedVia }) => ({
        type: `${boundary.type}-update`,
        path: boundary.url,
        acceptedPath: acceptedVia.url,
      }))
    );
    ws.send({
      type: "update",
      updates,
    });
  }
}

function updateModules(file, modules, { ws }) {}
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 === `'` || char === `"`) {
          prevState = state;
          state = LexerState.inQuoteString;
        }
        break;
      case LexerState.inQuoteString:
        if (char === `'` || char === `"`) {
          addDep(i);
          return false;
        } else {
          currentDep += char;
        }
        break;
      default:
        break;
    }
  }
  return false;
}
exports.handleHMRUpdate = handleHMRUpdate;
exports.updateModules = updateModules;
exports.lexAcceptedHmrDeps = lexAcceptedHmrDeps;
```

12.4 transformRequest.js

lib\server\transformRequest.js

```
const fs = require('fs-extra');
+const { parse } = require("url");
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 {
+   let fsPath = parse(id).pathname;
+   code = await fs.readFile(fsPath, 'utf-8')
  }
+ await server.moduleGraph.ensureEntryFromUrl(url)
  const transformResult = await pluginContainer.transform(code, id)
  return transformResult;
}
module.exports = transformRequest;
```

12.5 moduleGraph.js

lib\server\moduleGraph.js

```
class ModuleNode {

  importers = new Set();

  acceptedHmrDeps = new Set();
  constructor(url) {
    this.url = url;
    this.type = "js";
  }
}
class ModuleGraph {
  constructor(resolveId) {
    this.resolveId = resolveId;
  }
  idToModuleMap = new Map();

  getModuleById(id) {
    return this.idToModuleMap.get(id);
  }

  async ensureEntryFromUrl(rawUrl) {
    const [url, resolvedId] = await this.resolveUrl(rawUrl);
    let mod = this.idToModuleMap.get(resolvedId);
    if (!mod) {
      this.idToModuleMap.set(resolvedId, new ModuleNode(url));
    }
    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;
```

12.6 importAnalysis.js

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;
```

12.7 index.html

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 type="module" src="/src/main.js">script>
    [illegible]
  [illegible]>
[illegible]>
```

12.8 main.js

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();
  });
}
```

12.9 render.js

src\render.js

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

12.10 client.js

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}`);
}
```