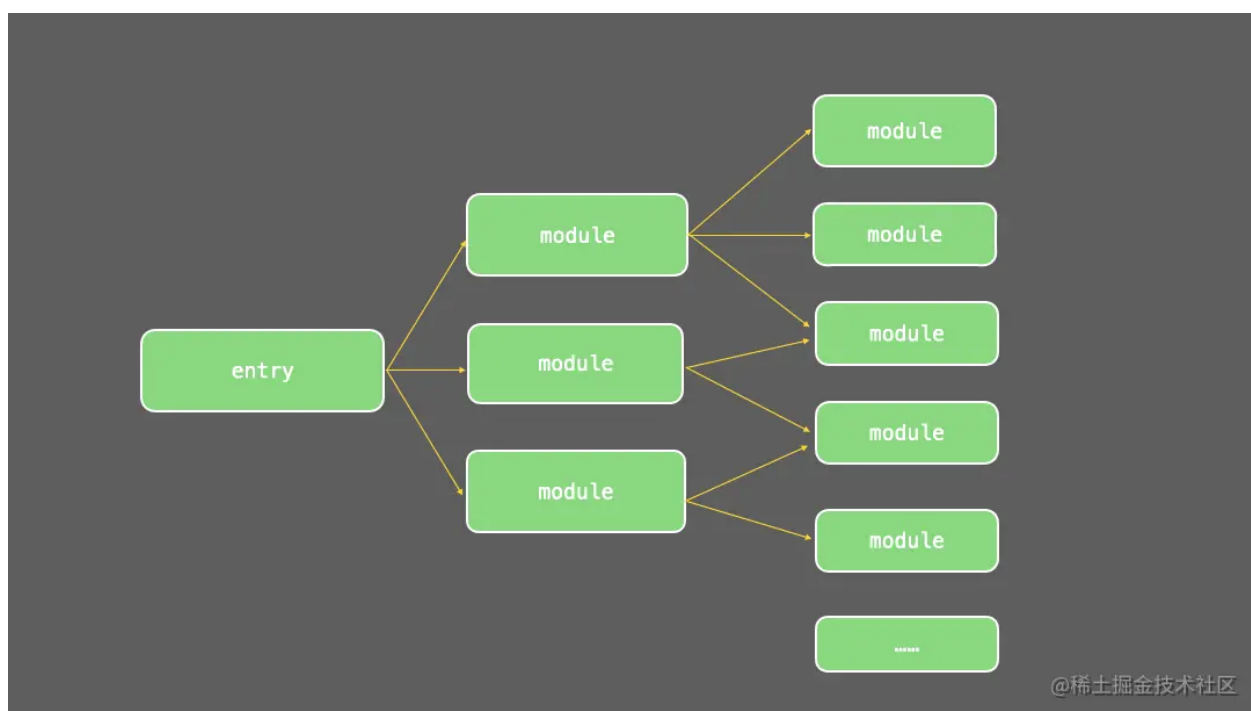


热更新：基于 ESM 的毫秒级 HMR 的实现揭秘

发布于 2022-05-09

在 [第 13 小节](#) 中，我们学习过 Vite 中 HMR 的 API 使用，同时也介绍了基于 HMR Boundary (HMR 边界) 的更新模式，即当一个模块发生变动时，Vite 会自动寻找更新边界，然后更新边界模块，如下图所示：



那么，在 Vite 内部，服务端究竟是如何定位到 HMR 边界模块，以及客户端是如何接受更新并加载最新模块内容的呢？

接下来的内容中，我就来和你一起深入 Vite 的底层实现，梳理 HMR 的各个实现要点，让你对 Vite 的 HMR 实现原理有比较深入的认识。

创建模块依赖图

为了方便管理各个模块之间的依赖关系，Vite 在 Dev Server 中创建了模块依赖图的数据结构，即 `ModuleGraph` 类，[点击查看实现源码](#)，Vite 中 HMR 边界模块的判定会依靠这个类来实现。

接下来，我们从以下几个维度看看这个图结构的创建过程。**创建依赖图**主要分为三个步骤：

- 初始化依赖图实例
- 创建依赖图节点
- 绑定各个模块节点的依赖关系

首先，Vite 在 Dev Server 启动时会初始化 ModuleGraph 的实例：

```
// packages/vite/src/node/server/index.ts
const moduleGraph: ModuleGraph = new ModuleGraph((url) =>
  container.resolveId(url)
);
```

接下来我们具体查看 **ModuleGraph** 这个类的实现。其中定义了若干个 Map，用来记录模块信息：

```
// 由原始请求 url 到模块节点的映射，如 /src/index.tsx
urlToModuleMap = new Map<string, ModuleNode>()
// 由模块 id 到模块节点的映射，其中 id 与原始请求 url，为经过 resolveId 钩子解析后的结果
idToModuleMap = new Map<string, ModuleNode>()
// 由文件到模块节点的映射，由于单文件可能包含多个模块，如 .vue 文件，因此 Map 的 value 值为一个集合
fileToModulesMap = new Map<string, Set<ModuleNode>>()
```

ModuleNode 对象即代表模块节点的具体信息，我们可以来看看它的数据结构：

```
class ModuleNode {
  // 原始请求 url
  url: string
  // 文件绝对路径 + query
  id: string | null = null
  // 文件绝对路径
  file: string | null = null
  type: 'js' | 'css'
  info?: ModuleInfo
  // resolveId 钩子返回结果中的元数据
  meta?: Record<string, any>
  // 该模块的引用方
  importers = new Set<ModuleNode>()
  // 该模块所依赖的模块
  importedModules = new Set<ModuleNode>()
  // 接受更新的模块
  acceptedHmrDeps = new Set<ModuleNode>()
  // 是否为`接受自身模块`的更新
  isSelfAccepting = false
  // 经过 transform 钩子后的编译结果
```

```

transformResult: TransformResult | null = null
// SSR 过程中经过 transform 钩子后的编译结果
ssrTransformResult: TransformResult | null = null
// SSR 过程中的模块信息
ssrModule: Record<string, any> | null = null
// 上一次热更新的时间戳
lastHMRTimestamp = 0

constructor(url: string) {
  this.url = url
  this.type = isDirectCSSRequest(url) ? 'css' : 'js'
}
}

```

ModuleNode 中包含的信息比较多，你需要重点关注的是 `importers` 和 `importedModules`，这两条信息分别代表了当前模块被哪些模块引用以及它依赖了哪些模块，是构建整个模块依赖图的根基所在。

那么，Vite 是在什么时候创建 ModuleNode 节点的呢？我们可以到 Vite Dev Server 中的 `transform` 中间件一探究竟：

```

// packages/vite/src/node/server/middlewares/transform.ts
// 核心转换逻辑
const result = await transformRequest(url, server, {
  html: req.headers.accept?.includes('text/html')
})

```

可以看到，`transform` 中间件的主要逻辑是调用 `transformRequest` 方法，我们来进一步查看这个方法的核心代码实现：

```

// packages/vite/src/node/server/transformRequest.ts
// 从 ModuleGraph 查找模块节点信息
const module = await server.moduleGraph.getModuleByUrl(url)
// 如果有则命中缓存
const cached =
  module && (ssr ? module.ssrTransformResult : module.transformResult)
if (cached) {
  return cached
}
// 否则调用 PluginContainer 的 resolveId 和 load 方法对进行模块加载
const id = (await pluginContainer.resolveId(url))?.id || url
const loadResult = await pluginContainer.load(id, { ssr })
// 然后通过调用 ensureEntryFromUrl 方法创建 ModuleNode
const mod = await moduleGraph.ensureEntryFromUrl(url)

```

接着我们看看 `ensureEntryFromUrl` 方法如何创建新的 ModuleNode 节点：

```

async ensureEntryFromUrl(rawUrl: string): Promise<ModuleNode> {
  // 实质是调用各个插件的 resolveId 钩子得到路径信息
  const [url, resolvedId, meta] = await this.resolveUrl(rawUrl)
  let mod = this.urlToModuleMap.get(url)
  if (!mod) {
    // 如果没有缓存，就创建新的 ModuleNode 对象
    // 并记录到 urlToModuleMap、idToModuleMap、fileToModulesMap 这三张表中
    mod = new ModuleNode(url)
    if (meta) mod.meta = meta
    this.urlToModuleMap.set(url, mod)
    mod.id = resolvedId
    this.idToModuleMap.set(resolvedId, mod)
    const file = (mod.file = cleanUrl(resolvedId))
    let fileMappedModules = this.fileToModulesMap.get(file)
    if (!fileMappedModules) {
      fileMappedModules = new Set()
      this.fileToModulesMap.set(file, fileMappedModules)
    }
    fileMappedModules.add(mod)
  }
  return mod
}

```

现在你应该明白了模块依赖图中各个 ModuleNode 节点是如何创建出来的，那么，各个节点的依赖关系是在什么时候绑定的呢？

我们不妨把目光集中到 `vite:import-analysis` 插件当中，在这个插件的 transform 钩子中，会对模块代码中的 import 语句进行分析，得到如下的一些信息：

- `importedUrls`：当前模块的依赖模块 url 集合。
- `acceptedUrls`：当前模块中通过 `import.meta.hot.accept` 声明的依赖模块 url 集合。
- `isSelfAccepting`：分析 `import.meta.hot.accept` 的用法，标记是否为接受自身更新的类型。

接下来会进入核心的 `模块依赖关系绑定` 的环节，核心代码如下：

```

// 引用方模块
const importerModule = moduleGraph.getModuleById(importer)
await moduleGraph.updateModuleInfo(
  importerModule,
  importedUrls,
  normalizedAcceptedUrls,
  isSelfAccepting
)

```

可以看到，绑定依赖关系的逻辑主要由 `ModuleGraph` 对象的 `updateModuleInfo` 方法实现，核心代码如下：

```
async updateModuleInfo(  
  mod: ModuleNode,  
  importedModules: Set<string | ModuleNode>,  
  acceptedModules: Set<string | ModuleNode>,  
  isSelfAccepting: boolean  
) {  
  mod.isSelfAccepting = isSelfAccepting  
  mod.importedModules = new Set()  
  // 绑定节点依赖关系  
  for (const imported of importedModules) {  
    const dep =  
      typeof imported === 'string'  
        ? await this.ensureEntryFromUrl(imported)  
        : imported  
    dep.importers.add(mod)  
    mod.importedModules.add(dep)  
  }  
  
  // 更新 acceptHmrDeps 信息  
  const deps = (mod.acceptedHmrDeps = new Set())  
  for (const accepted of acceptedModules) {  
    const dep =  
      typeof accepted === 'string'  
        ? await this.ensureEntryFromUrl(accepted)  
        : accepted  
    deps.add(dep)  
  }  
}
```

至此，模块间的依赖关系就成功进行绑定了。随着越来越多的模块经过 `vite:import-analysis` 的 transform 钩子处理，所有模块之间的依赖关系会被记录下来，整个依赖图的信息也就被补充完整了。

服务端收集更新模块

刚才我们分析了模块依赖图的实现，接下来再看看 Vite 服务端如何根据这个图结构收集更新模块。

首先，Vite 在服务启动时会通过 `chokidar` 新建文件监听器：

```
// packages/vite/src/node/server/index.ts  
import chokidar from 'chokidar'  
  
// 监听根目录下的文件  
const watcher = chokidar.watch(path.resolve(root));
```

```
// 修改文件
watcher.on('change', async (file) => {
  file = normalizePath(file)
  moduleGraph.onFileChange(file)
  await handleHMRUpdate(file, server)
})
// 新增文件
watcher.on('add', (file) => {
  handleFileAddUnlink(normalizePath(file), server)
})
// 删除文件
watcher.on('unlink', (file) => {
  handleFileAddUnlink(normalizePath(file), server, true)
})
```

然后，我们分别以修改文件、新增文件和删除文件这几个方面来介绍 HMR 在服务端的逻辑。

1. 修改文件

当业务代码中某个文件被修改时，Vite 首先会调用 `moduleGraph` 的 `onFileChange` 对模块图中的对应节点进行 **清除缓存** 的操作：

```
class ModuleGraph {
  onFileChange(file: string): void {
    const mods = this.getModulesByFile(file)
    if (mods) {
      const seen = new Set<ModuleNode>()
      // 将模块的缓存信息去除
      mods.forEach((mod) => {
        this.invalidateModule(mod, seen)
      })
    }
  }

  invalidateModule(mod: ModuleNode, seen: Set<ModuleNode> = new Set()): void {
    mod.info = undefined
    mod.transformResult = null
    mod.ssrTransformResult = null
  }
}
```

然后正式进入 HMR 收集更新的阶段，主要逻辑在 `handleHMRUpdate` 函数中，代码简化后如下：

```
// packages/vite/src/node/server/hmr.ts
export async function handleHMRUpdate(
  file: string,
```

```

server: ViteDevServer
): Promise<any> {
  const { ws, config, moduleGraph } = server
  const shortFile = getShortName(file, config.root)

  // 1. 配置文件/环境变量声明文件变化，直接重启服务
  // 代码省略

  // 2. 客户端注入的文件(vite/dist/client/client.mjs)更改
  // 给客户端发送 full-reload 信号，使之刷新页面
  if (file.startsWith(normalizedClientDir)) {
    ws.send({
      type: 'full-reload',
      path: '*'
    })
    return
  }
  // 3. 普通文件变动
  // 获取需要更新的模块
  const mods = moduleGraph.getModulesByFile(file)
  const timestamp = Date.now()
  // 初始化 HMR 上下文对象
  const hmrContext: HmrContext = {
    file,
    timestamp,
    modules: mods ? [...mods] : [],
    read: () => readModifiedFile(file),
    server
  }
  // 依次执行插件的 handleHotUpdate 钩子，拿到插件处理后的 HMR 模块
  for (const plugin of config.plugins) {
    if (plugin.handleHotUpdate) {
      const filteredModules = await plugin.handleHotUpdate(hmrContext)
      if (filteredModules) {
        hmrContext.modules = filteredModules
      }
    }
  }
  // updateModules—核心处理逻辑
  updateModules(shortFile, hmrContext.modules, timestamp, server)
}

```

从中可以看到，Vite 对于不同类型的文件，热更新的策略有所不同：

- 对于配置文件和环境变量声明文件的改动，Vite 会直接重启服务器。
- 对于客户端注入的文件(vite/dist/client/client.mjs)的改动，Vite 会给客户端发送 **full-reload** 信号，让客户端刷新页面。
- 对于普通文件改动，Vite 首先会获取需要热更新的模块，然后对这些模块依次查找热更新边界，然后将模块更新的信息传给客户端。

其中，对于普通文件的热更新边界查找的逻辑，主要集中在 **updateModules** 函数中，让我们来看看具体的实现：

```

function updateModules(
  file: string,
  modules: ModuleNode[],
  timestamp: number,
  { config, ws }: ViteDevServer
) {
  const updates: Update[] = []
  const invalidatedModules = new Set<ModuleNode>()
  let needFullReload = false
  // 遍历需要热更新的模块
  for (const mod of modules) {
    invalidate(mod, timestamp, invalidatedModules)
    if (needFullReload) {
      continue
    }
    // 初始化热更新边界集合
    const boundaries = new Set<{
      boundary: ModuleNode
      acceptedVia: ModuleNode
    }>()
    // 调用 propagateUpdate 函数，收集热更新边界
    const hasDeadEnd = propagateUpdate(mod, boundaries)
    // 返回值为 true 表示需要刷新页面，否则局部热更新即可
    if (hasDeadEnd) {
      needFullReload = true
      continue
    }
    // 记录热更新边界信息
    updates.push(
      ...[...boundaries].map(({ boundary, acceptedVia }) => ({
        type: `${boundary.type}-update` as Update['type'],
        timestamp,
        path: boundary.url,
        acceptedPath: acceptedVia.url
      })))
  }
}
// 如果被打上 full-reload 标识，则让客户端强制刷新页面
if (needFullReload) {
  ws.send({
    type: 'full-reload'
  })
} else {
  config.logger.info(
    updates
      .map(({ path }) => chalk.green(`hmr update `) + chalk.dim(path))
      .join('\n'),
    { clear: true, timestamp: true }
  )
  ws.send({
    type: 'update',
    updates
  })
}
}

```

// 热更新边界收集


```

function propagateUpdate(
  node: ModuleNode,
  boundaries: Set<{
    boundary: ModuleNode
    acceptedVia: ModuleNode
  }>,
  currentChain: ModuleNode[] = [node]
): boolean {
  // 接受自身模块更新
  if (node.isSelfAccepting) {
    boundaries.add({
      boundary: node,
      acceptedVia: node
    })
    return false
  }
  // 入口模块
  if (!node.importers.size) {
    return true
  }
  // 遍历引用方
  for (const importer of node.importers) {
    const subChain = currentChain.concat(importer)
    // 如果某个引用方模块接受了当前模块的更新
    // 那么将这个引用方模块作为热更新的边界
    if (importer.acceptedHmrDeps.has(node)) {
      boundaries.add({
        boundary: importer,
        acceptedVia: node
      })
      continue
    }

    if (currentChain.includes(importer)) {
      // 出现循环依赖，需要强制刷新页面
      return true
    }
    // 递归向更上层的引用方寻找热更新边界
    if (propagateUpdate(importer, boundaries, subChain)) {
      return true
    }
  }
  return false
}

```

可以看到，当热更新边界的信息收集完成后，服务端会将这些信息推送给客户端，从而完成局部的模块更新。

2. 新增和删除文件

对于新增和删除文件，Vite 也通过 `chokidar` 监听了相应的事件：

```

watcher.on('add', (file) => {
  handleFileAddUnlink(normalizePath(file), server)
})

watcher.on('unlink', (file) => {
  handleFileAddUnlink(normalizePath(file), server, true)
})

```

接下来，我们就来浏览一下 `handleFileAddUnlink` 的逻辑，代码简化后如下：

```

export async function handleFileAddUnlink(
  file: string,
  server: ViteDevServer,
  isUnlink = false
): Promise<void> {
  const modules = [...(server.moduleGraph.getModulesByFile(file) ?? [])]

  if (modules.length > 0) {
    updateModules(
      getShortName(file, server.config.root),
      modules,
      Date.now(),
      server
    )
  }
}

```

不难发现，这个函数同样是调用 `updateModules` 完成模块热更新边界的查找和更新信息的推送，而 `updateModules` 在上文中已经分析过，这里就不再赘述了。

客户端派发更新

好，从前面的内容中，我们知道，服务端会监听文件的改动，然后计算出对应的热更新信息，通过 WebSocket 将更新信息传递给客户端，具体来说，会给客户端发送如下的数据：

```

{
  type: "update",
  update: [
    {
      // 更新类型，也可能是 `css-update`
      type: "js-update",
      // 更新时间戳
      timestamp: 1650702020986,
      // 热更模块路径
      path: "/src/main.ts",
      // 接受的子模块路径
    }
  ]
}

```

```

    acceptedPath: "/src/render.ts"
  }
]
}
// 或者 full-reload 信号
{
  type: "full-reload"
}

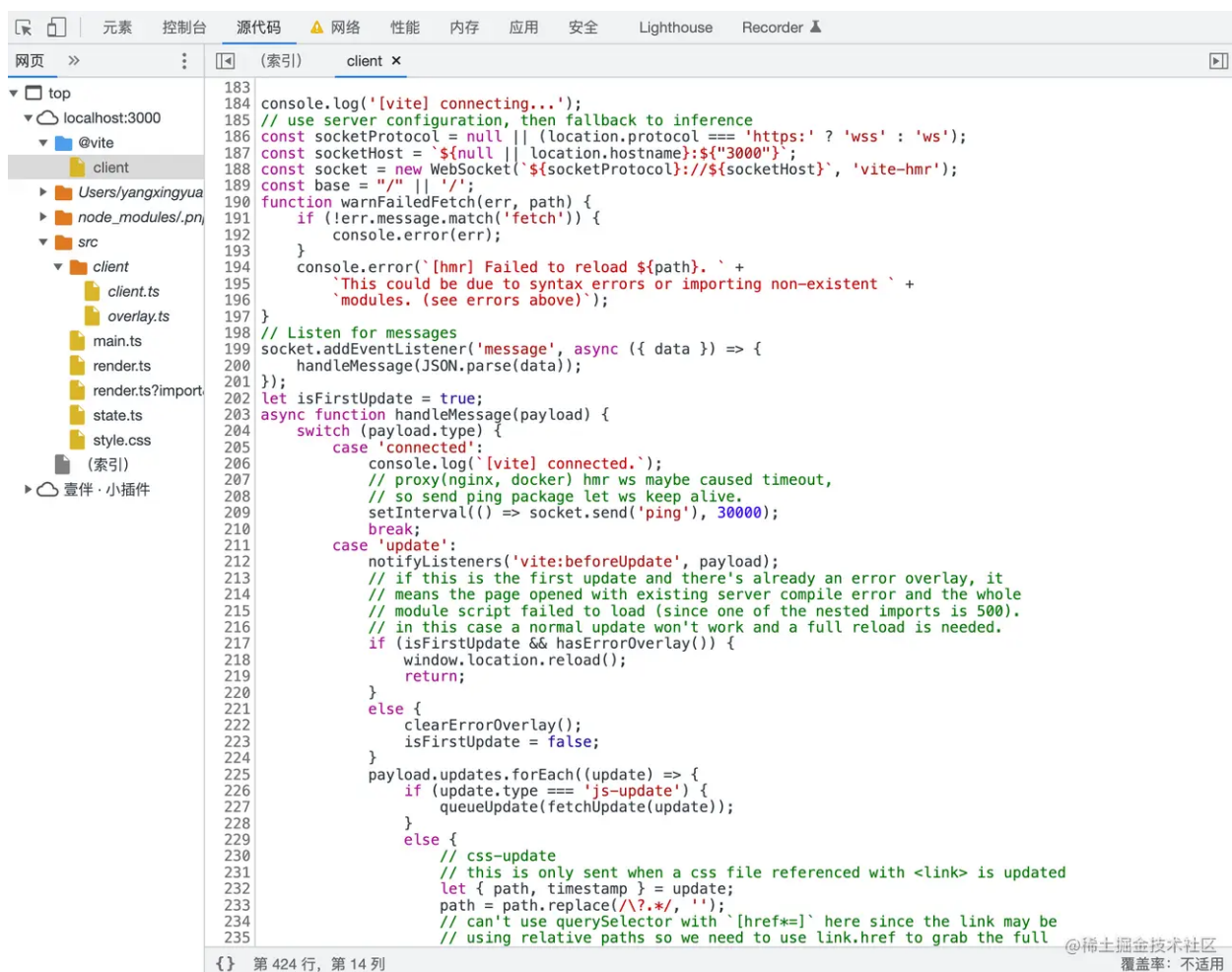
```

那么客户端是如何接受这些信息并进行模块更新的呢？

从上一节我们知道，Vite 在开发阶段会默认在 HTML 中注入一段客户端的脚本，即：

```
<script type="module" src="@vite/client"></script>
```

在启动任意一个 Vite 项目后，我们可以在浏览器查看具体的脚本内容：



从中你可以发现，客户端的脚本中创建了 WebSocket 客户端，并与 Vite Dev Server 中的 WebSocket 服务端([点击查看实现](#))建立双向连接：

```

const socketProtocol = null || (location.protocol === 'https:' ? 'wss' : 'ws');
const socketHost = `${null || location.hostname}:${"3000"}`;
const socket = new WebSocket(`${socketProtocol}://${socketHost}`, 'vite-hmr');

```

随后会监听 socket 实例的 `message` 事件，接收到服务端传来的更新信息：

```
socket.addEventListener('message', async ({ data }) => {
  handleMessage(JSON.parse(data));
});
```

接下来让我们把目光集中在 `handleMessage` 函数中：

```
async function handleMessage(payload: HMRPayload) {
  switch (payload.type) {
    case 'connected':
      console.log(`[vite] connected.`)
      // 心跳检测
      setInterval(() => socket.send('ping'), __HMR_TIMEOUT__)
      break
    case 'update':
      payload.updates.forEach((update) => {
        if (update.type === 'js-update') {
          queueUpdate(fetchUpdate(update))
        } else {
          // css-update
          // 省略实现
          console.log(`[vite] css hot updated: ${path}`)
        }
      })
      break
    case 'full-reload':
      // 刷新页面
      location.reload()
      // 省略其它消息类型
  }
}
```

其中，我们重点关注 js 的更新逻辑，即下面这行代码：

```
queueUpdate(fetchUpdate(update))
```

到底做了些什么。

我们先来看看 `queueUpdate` 和 `fetchUpdate` 这两个函数的实现：

```
let pending = false
let queued: Promise<() => void> | undefined[] = []

// 批量任务处理，不与具体的热更新行为挂钩，主要起任务调度作用
async function queueUpdate(p: Promise<() => void> | undefined) {
  queued.push(p)
```

```

if (!pending) {
  pending = true
  await Promise.resolve()
  pending = false
  const loading = [...queued]
  queued = []
  ;(await Promise.all(loading)).forEach((fn) => fn && fn())
}
}

```

// 派发热更新的主要逻辑

```

async function fetchUpdate({ path, acceptedPath, timestamp }: Update) {
  // 后文会介绍 hotModuleMap 的作用，你暂且不用纠结实现，可以理解为 HMR 边界模块相关的信息
  const mod = hotModulesMap.get(path)
  const moduleMap = new Map()
  const isSelfUpdate = path === acceptedPath

  // 1. 整理需要更新的模块集合
  const modulesToUpdate = new Set<string>()
  if (isSelfUpdate) {
    // 接受自身更新
    modulesToUpdate.add(path)
  } else {
    // 接受子模块更新
    for (const { deps } of mod.callbacks) {
      deps.forEach((dep) => {
        if (acceptedPath === dep) {
          modulesToUpdate.add(dep)
        }
      })
    }
  }

  // 2. 整理需要执行的更新回调函数
  // 注: mod.callbacks 为 import.meta.hot.accept 中绑定的更新回调函数，后文会介绍
  const qualifiedCallbacks = mod.callbacks.filter(({ deps }) => {
    return deps.some((dep) => modulesToUpdate.has(dep))
  })

  // 3. 对将要更新的模块进行失活操作，并通过动态 import 拉取最新的模块信息
  await Promise.all(
    Array.from(modulesToUpdate).map(async (dep) => {
      const disposer = disposeMap.get(dep)
      if (disposer) await disposer(dataMap.get(dep))
      const [path, query] = dep.split('?')
      try {
        const newMod = await import(
          /* @vite-ignore */
          base +
            path.slice(1) +
            `?import&t=${timestamp}${query ? `&${query}` : ''}`
        )
        moduleMap.set(dep, newMod)
      } catch (e) {
        warnFailedFetch(e, dep)
      }
    })
  )

  // 4. 返回一个函数，用来执行所有的更新回调
  return () => {

```

```

    for (const { deps, fn } of qualifiedCallbacks) {
      fn(deps.map((dep) => moduleMap.get(dep)))
    }
    const loggedPath = isSelfUpdate ? path : `${acceptedPath} via ${path}`
    console.log(`[vite] hot updated: ${loggedPath}`)
  }
}

```

对热更新的边界模块来讲，我们需要在客户端获取这些信息：

- 边界模块所接受(accept)的模块
- accept 的模块触发更新后的回调

我们知道，在 `vite:import-analysis` 插件中，会给包含热更新逻辑的模块注入一些工具代码，如下图所示：

```

1 import {createHotContext as __vite__createHotContext} from "@vite/client";
2 import.meta.hot = __vite__createHotContext("/src/main.ts");
3
4 import {render} from "/src/render.ts";
5 import {initState} from "/src/state.ts";
6 render();
7 initState();
8 if (import.meta.hot) {
9   import.meta.hot.accept(["/src/render.ts", "/src/state.ts"], (modules)=>{
10     const [renderModule,stateModule] = modules;
11     if (renderModule) {
12       renderModule.render();
13     }
14     if (stateModule) {
15       stateModule.initState();
16     }
17   });
18 }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2
```

```

    accept(deps: any, callback?: any) {
      if (typeof deps === 'function' || !deps) {
        acceptDeps([ownerPath], ([mod]) => deps && deps(mod))
      } else if (typeof deps === 'string') {
        acceptDeps([deps], ([mod]) => callback && callback(mod))
      } else if (Array.isArray(deps)) {
        acceptDeps(deps, callback)
      } else {
        throw new Error(`invalid hot.accept() usage.`)
      }
    },
    // import.meta.hot.dispose
    // import.meta.hot.invalidate
    // 省略更多方法的实现
  }
}

```

因此，Vite 给每个热更新边界模块注入的工具代码主要有两个作用：

- 注入 `import.meta.hot` 对象的实现
- 将当前模块 `accept` 过的模块和更新回调函数记录到 `hotModulesMap` 表中

而前面所说的 `fetchUpdate` 函数则是通过 `hotModuleMap` 来获取边界模块的相关信息，在 `accept` 的模块发生变动后，通过动态 `import` 拉取最新的模块内容，然后返回更新回调，让 `queueUpdate` 这个调度函数执行更新回调，从而完成**派发更新**的过程。至此，HMR 的过程就结束了。

小结

好，本小节的内容就到这里。你需要重点掌握 Vite 中的**模块依赖图实现**、**服务端收集更新模块**和**客户端派发更新**的原理。

首先，Vite 为了更方便地管理模块之间的关系，创建了模块依赖图的数据结构，在 HMR 过程中，服务端会根据这张图来寻找 HMR 边界模块。

其次，HMR 更新由客户端和服务端配合完成，两者通过 `WebSocket` 进行数据传输。在服务端，Vite 通过查找模块依赖图确定热更新的边界，并将局部更新的信息传递给客户端，而客户端接收到热更信息后，会通过动态 `import` 请求并加载最新模块的内容，并执行派发更新的回调，即 `import.meta.hot.accept` 中定义的回调函数，从而完成完整的热更新过程。

最后，欢迎你在评论区记录本节的学习心得，也恭喜你完成了**源码精读章节**的学习，让我们下一章再见！

上一篇：插件流水线：从整体到局部，理解 Vite 的核心编译能力

下一篇：手写 Vite: 实现 no-bundle 开发服务(上)