

Webpack 原理系列十：HMR 原理全解析

原创 范文杰 Tecvan 2021-09-13 09:55

收录于合集

#Webpack

18个

这是 Webpack 原理分析系列第十篇文章，前文可到公众号【Tecvan】查阅。



Tecvan

All or nothing, now or never 🙌

48篇原创内容

公众号

一、什么是 HMR

HMR 全称 Hot Module Replacement，中文语境通常翻译为模块热更新，它能够在保持页面状态的情况下动态替换资源模块，提供丝滑顺畅的 Web 页面开发体验。

WEBPACK HOT MODULE REPLACEMENT EXPLAINED

```
import component from "../component";

document.body.appendChild(component);
// Check if HMR interface is enabled
if (module.hot) {
  // Accept hot update
  module.hot.accept();
}
```

Tecvan

HMR 最初由 Webpack 设计实现，至今已几乎成为现代工程化工具必备特性之一。

1.1 HMR 之前

在 HMR 之前，应用的加载、更新是一种页面级别的原子操作，即使只是单个代码文件发生变更都需要刷新整个页面才能最新代码映射到浏览器上，这会丢失之前在页面执行过的所有交互与状态，例如：

- 对于复杂表单场景，这意味着你可能需要重新填充非常多字段信息
- 弹框消失，你必须重新执行交互动作才会重新弹出

再小的改动，例如更新字体大小，改变备注信息都会需要整个页面重新加载执行，影响开发体验。引入 HMR 后，虽然无法覆盖所有场景，但大多数小改动都可以实时热更新到页面上，从而确保连续、顺畅的开发调试体验，对开发效率有较大增益效果。

1.2 使用 HMR

Webpack 生态下，只需要经过简单的配置即可启动 HMR 功能，大致上分两步：

- 配置 `devServer.hot` 属性为 `true`，如：

```
// webpack.config.js
module.exports = {
  // ...
  devServer: {
    // 必须设置 devServer.hot = true, 启动 HMR 功能
    hot: true
  }
};
```

- 之后，还需要调用 `module.hot.accept` 接口，声明如何将模块安全地替换为最新代码，如：

```
import component from "./component";
let demoComponent = component();

document.body.appendChild(demoComponent);

// HMR interface
if (module.hot) {
  // Capture hot update
  module.hot.accept("./component", () => {
    const nextComponent = component();
```

```
// Replace old content with the hot loaded one
document.body.replaceChild(nextComponent, demoComponent);

demoComponent = nextComponent;
});
}
```

模块代码的替换逻辑可能非常复杂，幸运的是我们通常不太需要对此过多关注，因为业界许多 Webpack Loader 已经提供了针对不同资源的 HMR 功能，例如：

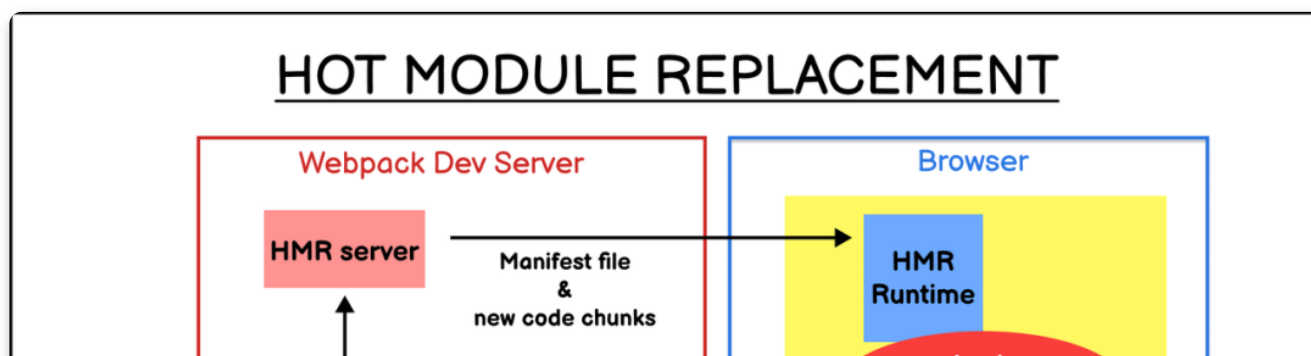
- `style-loader` 内置 Css 模块热更
- `vue-loader` 内置 Vue 模块热更
- `react-hot-reload` 内置 React 模块热更接口

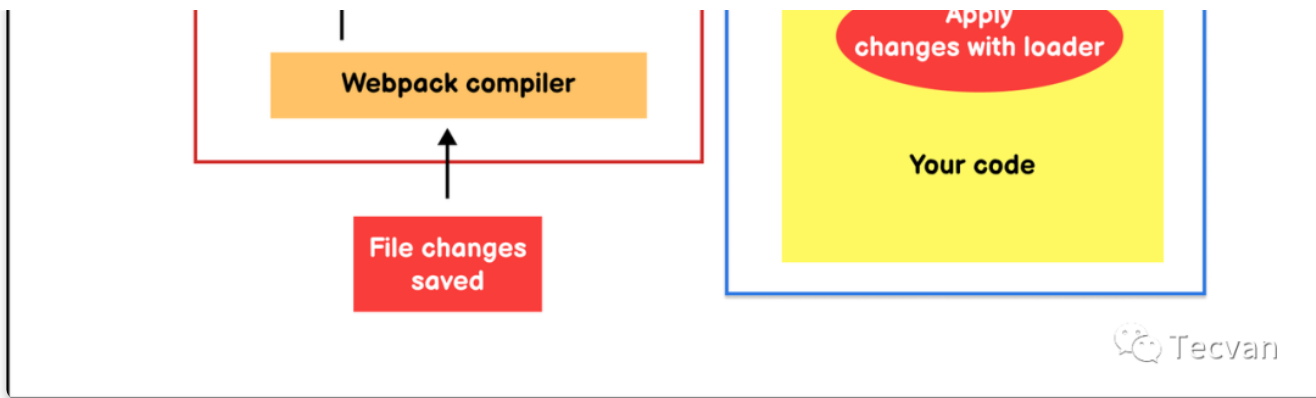
因此，站在使用的角度，只需要针对不同资源配置对应支持 HMR 的 Loader 即可，很容易上手。

二、实现原理

Webpack HMR 特性的原理并不复杂，核心流程：

1. 使用 `webpack-dev-server` (后面简称 WDS) 托管静态资源，同时以 Runtime 方式注入 HMR 客户端代码
2. 浏览器加载页面后，与 WDS 建立 WebSocket 连接
3. Webpack 监听到文件变化后，增量构建发生变更的模块，并通过 WebSocket 发送 `hash` 事件
4. 浏览器接收到 `hash` 事件后，请求 `manifest` 资源文件，确认增量变更范围
5. 浏览器加载发生变更的增量模块
6. Webpack 运行时触发变更模块的 `module.hot.accept` 回调，执行代码变更逻辑
7. done





接下来我会展开 HMR 的核心源码，详细讲解 Webpack 5 中 Hot Module Replacement 原理的关键部分，内容略微晦涩，不感兴趣的同学可以直接跳到下一章。

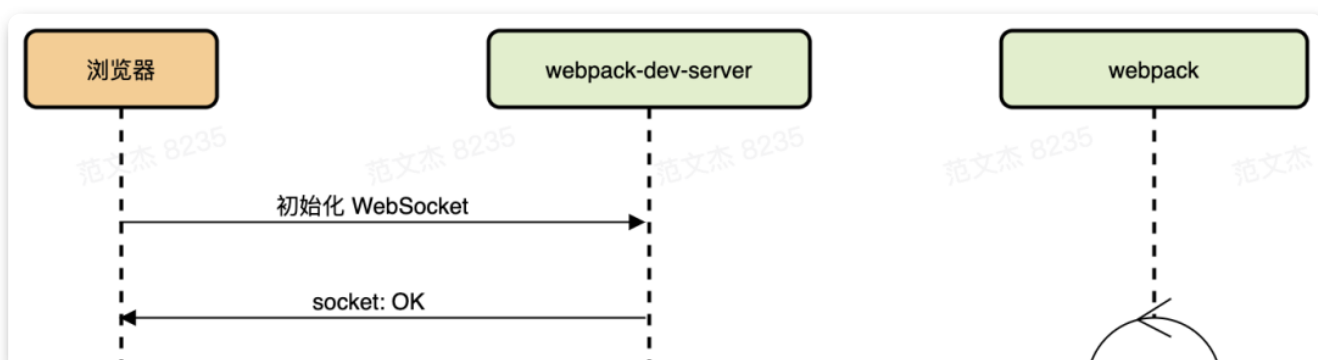
2.1 注入 HMR 客户端运行时

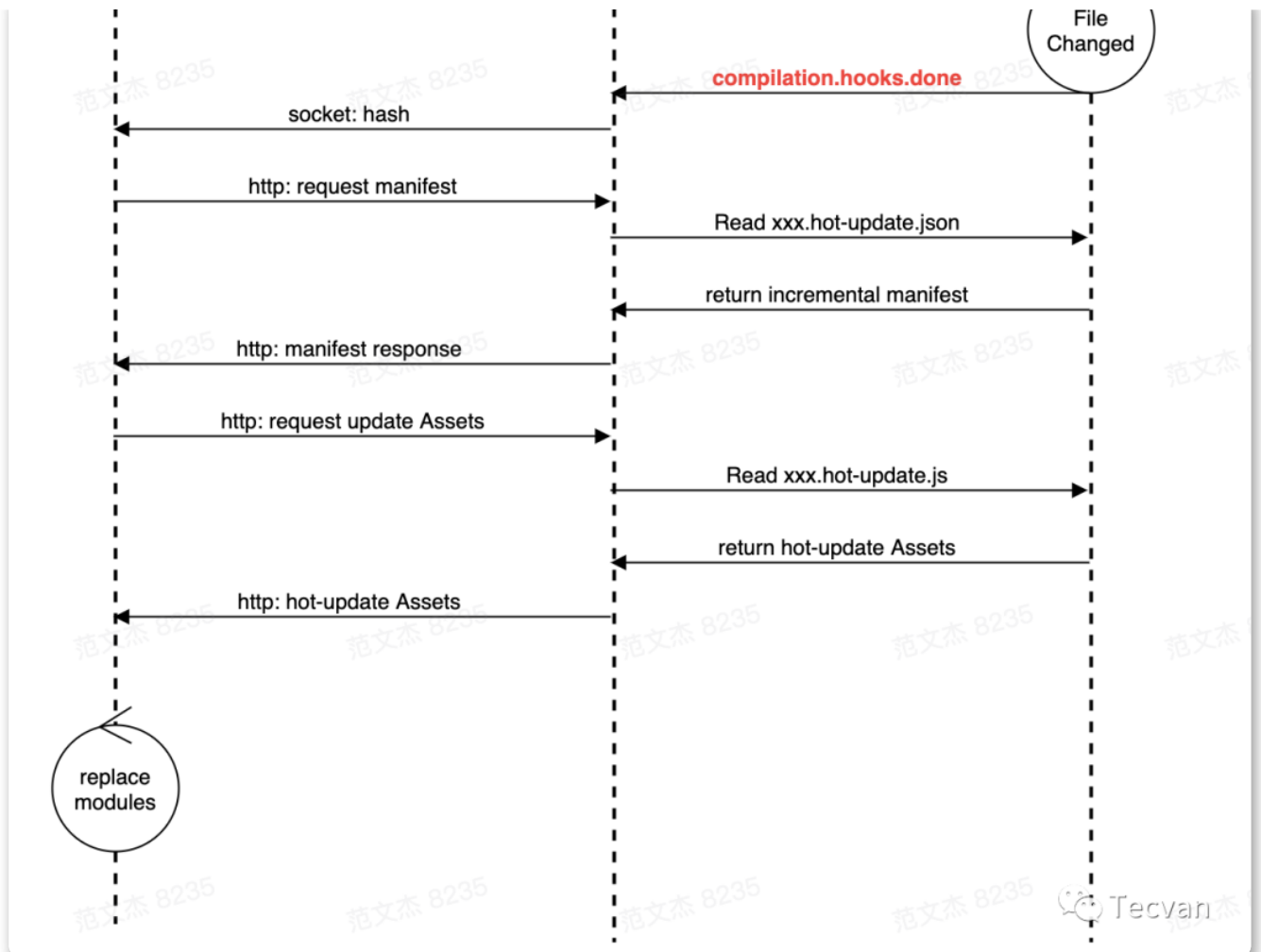
执行 `npx webpack serve` 命令后，WDS 调用 `HotModuleReplacementPlugin` 插件向应用的主 Chunk 注入一系列 HMR Runtime，包括：

- 用于建立 WebSocket 连接，处理 `hash` 等消息的运行时代码
- 用于加载热更新资源的 `RuntimeGlobals.hmrDownloadManifest` 与 `RuntimeGlobals.hmrDownloadUpdateHandlers` 接口
- 用于处理模块更新策略的 `module.hot.accept` 接口
- 等等

关于 Webpack Runtime，可参考 [Webpack 原理系列六：彻底理解 Webpack 运行时](#)。

经过 `HotModuleReplacementPlugin` 处理后，构建产物中即包含了所有运行 HMR 所需的客户端运行时与接口。这些 HMR 运行时会在浏览器执行一套基于 WebSocket 消息的时序框架，如图：





2.2 增量构建

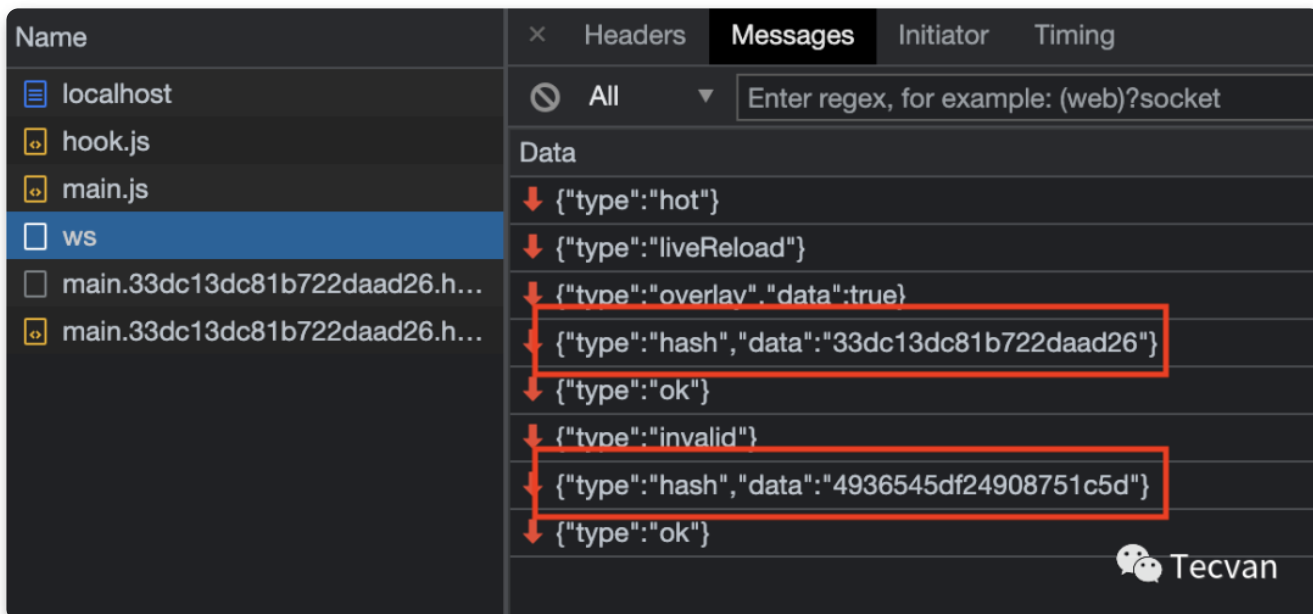
除注入客户端代码外，`HotModuleReplacementPlugin` 插件还会借助 Webpack 的 `watch` 能力，在代码文件发生变化后执行增量构建，生成：

- `manifest` 文件：JSON 格式文件，包含所有发生变更的模块列表，命名为 `[hash].hot-update.json`
- 模块变更文件：js 格式，包含编译后的模块代码，命名为 `[hash].hot-update.js`

增量构建完毕后，Webpack 将触发 `compilation.hooks.done` 钩子，并传递本次构建的统计信息对象 `stats`。WDS 则监听 `done` 钩子，在回调中通过 WebSocket 发送模块更新消息：

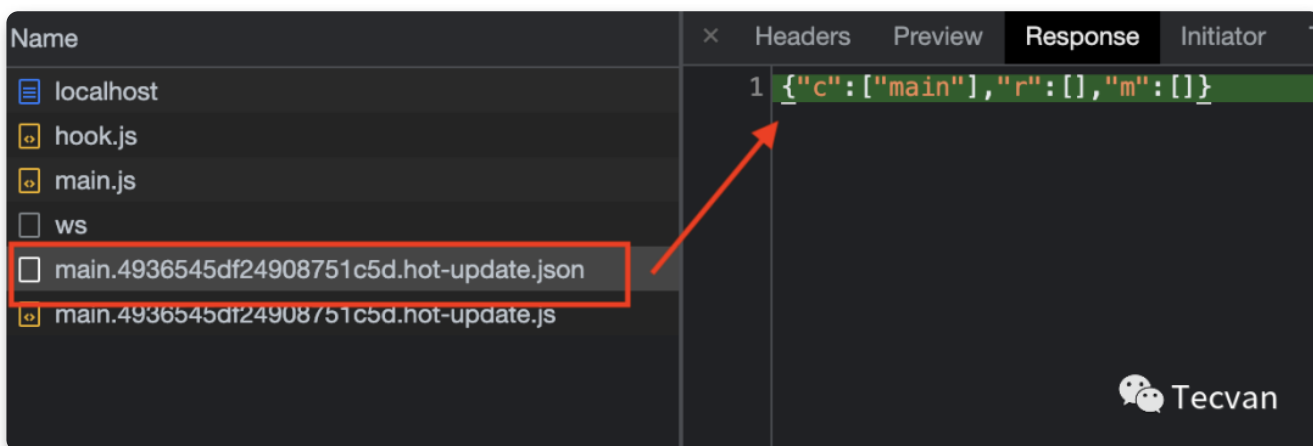
```
{"type":"hash","data":"${stats.hash}"}
```

实际效果：



2.3 加载更新

客户端接受到 `hash` 消息后，首先发出 `manifest` 请求获取本轮热更新涉及的 `chunk`，如：



注意，在 Webpack 4 及之前，热更新文件以模块为单位，即所有发生变化的模块都会生成对应的热更新文件；Webpack 5 之后热更新文件以 `chunk` 为单位，如上例中，`main` `chunk` 下任意文件的变化都只会生成 `main.[hash].hot-update.js` 更新文件。

`manifest` 请求完成后，客户端 HMR 运行时开始下载发生变化的 `chunk` 文件，将最新模块代码加载到本地。

2.4 module.hot.accept 回调

经过上述步骤，浏览器加载完最新模块代码后，HMR 运行时会继续触发 `module.hot.accept` 回调，将最新代码替换到运行环境中。

`module.hot.accept` 是 HMR 运行时暴露给用户代码的重要接口之一，它在 Webpack HMR 体系中开了一个口子，让用户能够自定义模块热替换的逻辑。

`module.hot.accept` 接口签名如下：

```
module.hot.accept(path?: string, callback?: function);
```

它接受两个参数：

- `path`：指定需要拦截变更行为的模块路径
- `callback`：模块更新后，将最新模块代码应用到运行环境的函数

例如，对于如下代码：

```
// src/bar.js
export const bar = 'bar'

// src/index.js
import { bar } from './bar';

const node = document.createElement('div')
node.innerText = bar;
document.body.appendChild(node)

module.hot.accept('./bar.js', function () {
  node.innerText = bar;
})
```

示例中，`module.hot.accept` 函数监听 `./bar.js` 模块的变更事件，一旦代码发生变动就触发回调，将 `./bar.js` 导出的值应用到页面上，从而实现热更新效果。

`module.hot.accept` 的作用并不复杂，但使用过程中还是有一些值得注意的点，下面细讲。

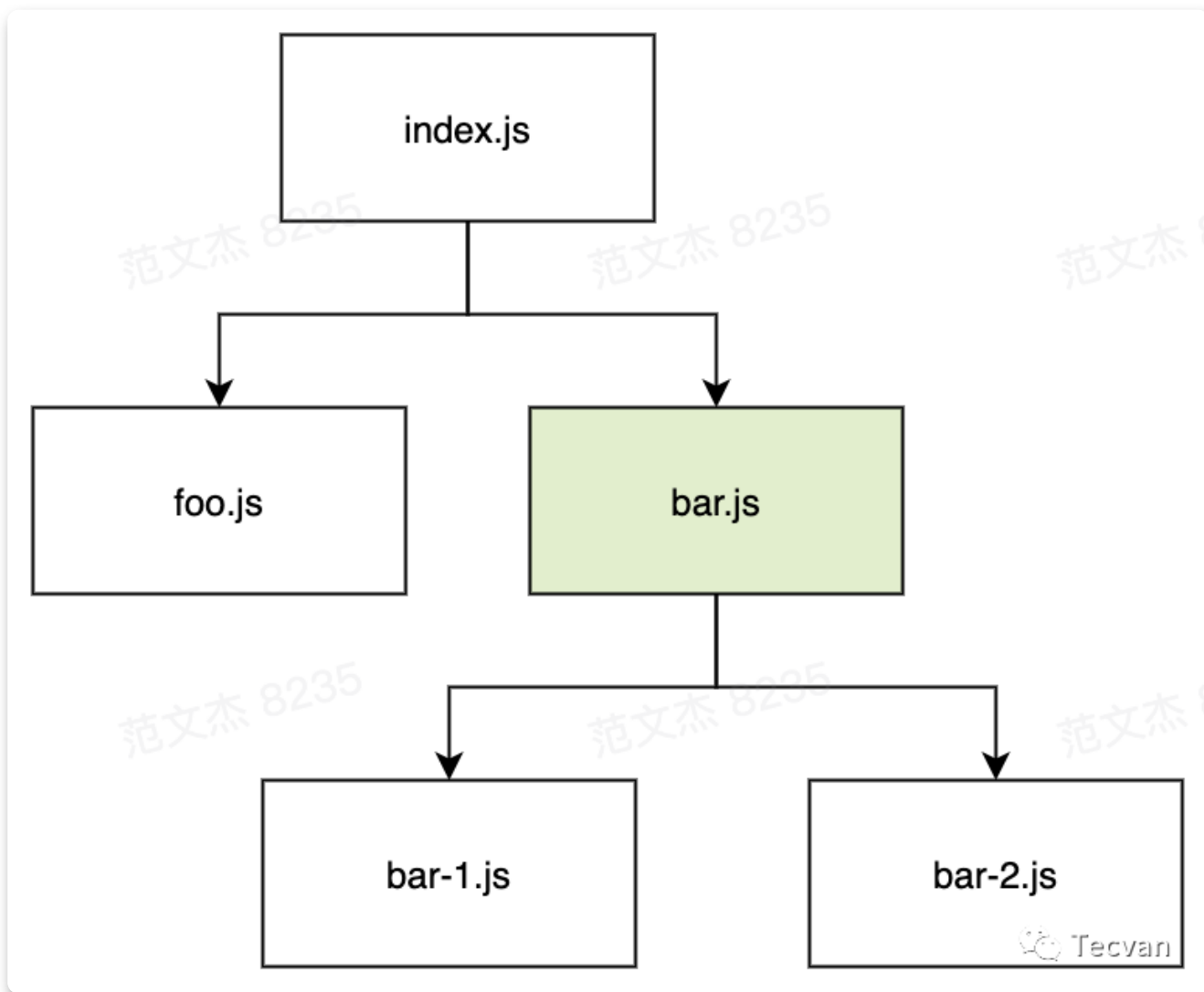
2.4.1 失败兜底

`module.hot.accept` 函数只接受具体路径的 `path` 参数，也就是说我们无法通过 `glob` 或类似风格的方式批量注册热更新回调。

一旦某个模块没有注册对应的 `module.hot.accept` 函数后，HMR 运行时会执行兜底策略，通常是刷新页面，确保页面上运行的始终是最新的代码。

2.4.2 更新事件冒泡

在 Webpack HMR 框架中，`module.hot.accept` 函数只能捕获当前模块对应子孙模块的更新事件，例如对于下面的模块依赖树：



示例中，更新事件会沿着模块依赖树自底向上逐级传递，从 `foo` 到 `index`，从 `bar-1` 到 `bar` 再到 `index`，但不支持反向或跨子树传递，也就是说：

- 在 `foo.js` 中无法捕获 `bar.js` 及其子模块的变更事件
- 在 `bar-1.js` 中无法捕获 `bar.js` 的变更事件

这一特性与 DOM 事件规范中的冒泡过程极为相似，使用时如果摸不准模块的依赖关系，建议直接在应用的入口文件中编写热更新函数。

2.4.3 无参数调用

除上述调用方式外，`module.hot.accept` 函数还支持无参数调用风格，作用是捕获当前文件的变更事件，并从模块第一行开始重新运行该模块的代码，例如：

```
// src/bar.js
console.log('bar');

module.hot.accept();
```

示例模块发生变动之后，会从头开始重复执行 `console.log` 语句。

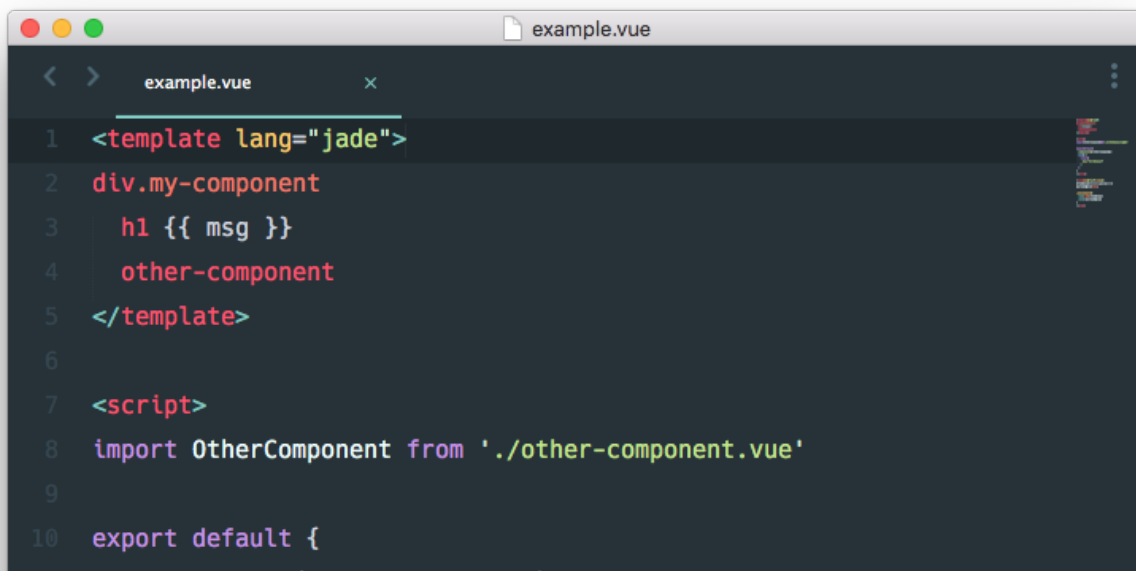
2.5 小结

回顾整个 HMR 过程，所有的状态流转均由 WebSocket 消息驱动，这部分逻辑由 HMR 运行时控制，开发者几乎无感。

唯一需要开发者关心的是为每一个需要处理热更新的文件注册 `module.hot.accept` 回调，所幸这部分需求已经被许多成熟的 Loader 处理，作为示例，下一节我们挖掘 `vue-loader` 源码，学习如何灵活使用 `module.hot.accept` 函数处理文件更新。

三、vue-loader 如何实现 HMR

`vue-loader` 是一个用于处理 Vue Single File Component 的 Webpack 加载器，它能够如下格式的内容转译为可在浏览器运行的等价代码：



```
example.vue
1 <template lang="jade">
2   div.my-component
3     h1 {{ msg }}
4     other-component
5 </template>
6
7 <script>
8   import OtherComponent from './other-component.vue'
9
10  export default {
```

```

11   components: { OtherComponent },
12   data() {
13     return {
14       msg: 'Hello Vue.js!'
15     }
16   }
17 }
18 </script>
19
20 <style lang="sass" scoped>
21 $font-stack: Helvetica, sans-serif;
22 $primary-color: #333;
23
24 .my-component {
25   font: 100% $font-stack;
26   color: $primary-color;
27 }
28 </style>

```

Line 1, Column 23 Spaces: 2 Vue Component



除常规的代码转译外，在 HMR 模式下，`vue-loader` 还会为每一个 Vue 文件注入一段处理模块替换的逻辑，如：

```

"./src/a.vue":
/*!*****!*\
  *** ./src/a.vue ***
  \*****/
/***/

((module, __webpack_exports__, __webpack_require__) => {
  // 模块代码
  // ...
  /* hot reload */
  if (true) {
    var api = __webpack_require__( /*! ../node_modules/vue-hot-reload-api/dist/index.js */ "../no
    api.install(__webpack_require__( /*! vue */ "../node_modules/vue/dist/vue.runtime.esm.js"))
    if (api.compatible) {
      module.hot.accept()
      if (!api.isRecorded('45c6ab58')) {

```

```

    api.createRecord('45c6ab58', component.options)
  } else {
    api.reload('45c6ab58', component.options)
  }
  module.hot.accept( /*! ./a.vue?vue&type=template&id=45c6ab58& */ " ./src/a.vue?vue&type=te
  /* harmony import */
  _a_vue_vue_type_template_id_45c6ab58___WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(
  (function () {
    api.rerender('45c6ab58', {
      render: _a_vue_vue_type_template_id_45c6ab58___WEBPACK_IMPORTED_MODULE_0__.render,
      staticRenderFns: _a_vue_vue_type_template_id_45c6ab58___WEBPACK_IMPORTED_MODULE_0__.s
    })
  })(__WEBPACK_IMPORTED_MODULE_0___);
  })
}
// ...

/***/
}),

```

这段被注入用于处理模块热替换的代码，主要步骤有：

- 首次执行时，调用 `api.createRecord` 记录组件配置，`api` 为 `vue-hot-reload-api` 库暴露的接口
- 执行 `module.hot.accept()` 语句，监听当前模块变更事件，当模块发生变化时调用 `api.reload`
- 执行 `module.hot.accept("xxx.vue?vue&type=template&xxx", fn)`，监听 Vue 文件 `template` 代码的变更事件，当 `template` 模块发生变更时调用 `api.rerender`

为什么需要调用两次 `module.hot.accept`？

这是因为 `vue-loader` 在做转译时，会将 SFC 不同板块拆解成多个 `module`，例如：`template` 对应生成 `xxx.vue?vue&type=template`；`script` 对应生成 `xxx.vue?vue&type=script`。因此，`vue-loader` 必须为这些不同的 `module` 分别调用 `accept` 接口，才能处理好不同代码块的变更事件。

可以看到，`vue-loader` 对 HMR 的支持，基本上围绕 `vue-hot-reload-api` 展开，当代码文件发生变化触发 `module.hot.accept` 回调时，会根据情况执行 `vue-hot-reload-api` 暴露的 `reload` 与 `rerender` 函数，两者最终都会触发组件实例的 `$forceUpdate` 函数强制执行重新渲染。

四、总结

最后再回顾一下，Webpack 的 HMR 特性有两个重点，一是监听文件变化并通过 WebSocket 发送变更消息；二是需要客户端提供配合，通过 `module.hot.accept` 接口明确告知 Webpack 如何执行代码替换。整体盘下来，并没有想象中那么困难。



Tecvan

All or nothing, now or never 🙌

48篇原创内容

公众号

最近很忙，双月OKR、年中绩效，还有一些突发事件，这篇文章实际上在上周就完成了80%，但一直没时间收尾，后面我尽量保持 1-2 周一更吧。

BTW，字节游戏中台-前端团队持续热招，欢迎直接联系我内推，我会跟进内推整个过程，**「知无不言言无不尽！」**

收录于合集 #Webpack 18

上一篇

Webpack 性能系列三：提升编译性能

下一篇

Webpack 原理系列九：Tree-Shaking 实现原理

阅读原文

喜欢此内容的人还喜欢

[科普] JS中Object的keys是无序的吗

Tecvan