训练营第三天 - 进阶与展望

> 尤雨溪: Vite 会取代 vue-cli 吗?
>
> https://juejin.cn/post/6935750217924870152
>
> 杨村长 Vite实战
>
> https://juejin.cn/post/6926822933721513998
>
> https://github.com/57code/vite2-in-action

一、Vite是什么

Vite(读音类似于[weɪt]，法语，快的意思) 是一个由原生 ES Module 驱动的 Web 开发构建工具。在开发环境下基于浏览器原生 ES imports 开发，在生产环境下基于 Rollup 打包

vite 的特点

- Lightning fast cold server start - 闪电般的冷启动速度
- Instant hot module replacement (HMR) - 即时热模块更换（热更新）
- True on-demand compilation - 真正的按需编译

要求

```
- Vite 要求项目完全由 ES Module 模块组成
- common.js 模块不能直接在 Vite 上使用
- 打包上依旧还是使用 rollup 等传统打包工具
```

安装

```
npm i vite -s
```

```
# 指定文件名和模板
npm init @vitejs/app vite-element-admin --template vue
```

Vite2主要变化

- 配置选项变化：`vue特有选项`、创建选项、css选项、jsx选项等、别名行为变化：不再要求/开头或结尾
- Vue支持：通过 @vitejs/plugin-vue插件支持
- React支持
- HMR API变化
- 清单格式变化
- 插件API重新设计

配置别名

vite.config.js

```
import path from 'path'
export default {
  resolve: {
    alias: {
      "/@": path.resolve(__dirname, "src"),
      comps: path.resolve(__dirname, "src/components"),
    },
  },
}
```

配置文件vite.config.js

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue()]  // 插件形式支持vue
})
```

添加路由

```
npm i vue-router@next -S
```

添加视图 /views/home.vue

```
<template>
    <div>Home....</div>
</template>
```

路由配置 /router/index.js

```
import { createRouter, createWebHashHistory } from 'vue-router';

const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    { path: '/', component: () => import('views/home.vue') }
  ]
});

export default router
```

引入 main.js

```
import router from "@/router";
createApp(App)
.use(router) // 添加路由插件
.mount("#app");
```

修改布局 main.js

```
<template>
  <router-view></router-view>
</template>
```

状态管理

```
npm i vuex@next -S
```

Store配置，`store/index.js`

```
import { createStore } from "vuex";

export default createStore({
  state: {
    count: 0,
  },
  mutations: {
    increment(state) {
      state.count++;
    },
  },
});
```

引入，`main.js`

```
import store from "@/store";
createApp(App).use(store).mount("#app");
```

使用状态 /views/home.vue

```
<div>{{ $store.state.count }}</div>
<button @click="$store.commit('increment')">Add</button>
```

二、Vite原理分析

1. EsModule

服务器端

```
const Koa = require('koa')
const app = new Koa()
app.use(async (ctx) => {
```

```
  const {
    request: { url, query },
  } = ctx;
  console.log("url:" + url, "query type", query.type);
  // 首页
  if (url == "/") {
    ctx.type = "text/html";
    let content = fs.readFileSync("./index.html", "utf-8");
    ctx.body = content;
  }
})
app.listen(3000, () => {
  console.log('Vite Start ....')
})
```

新建页面index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <link rel="icon" href="/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vite App</title>
</head>
<body>
  <h1>然叔 666</h1>
  <div id="app"></div>
  <script>
  </script>
  <script type="module" src="/src/main.js"></script>
</body>
</html>
```

新建/src/main.js

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

添加模块解析 /index.js

/src/moduleA

```
export const str = "Hello Vite";
```

/src/main.js

```
import { str } from "./moduelA.js";
console.log(str);
```

```
else if (url.endsWith(".js")) {
    // js文件
    const p = path.resolve(__dirname, url.slice(1));
    ctx.type = "application/javascript";
    const content = fs.readFileSync(p, "utf-8");
   ctx.body = content
  }
```

添加依赖解析

> From ('./xxxx') => from ('./xxx')
>
> From ('yyyy') => from ('/@modules/yyyy')

```
function rewriteImport(content) {
  return content.replace(/ from ['|"]([^'"]+)['|"]/g, function (s0, s1) {
    console.log("s", s0, s1);
    // . ../ /开头的，都是相对路径
    if (s1[0] !== "." && s1[1] !== "/") {
      return `from '/@modules/${s1}'`;
    } else {
      return s0;
    }
  });
}
// 添加模块改写
 ctx.body = rewriteImport(content);
```

第三方依赖支持

/src/main.js

```
import { createApp, h } from "vue";
const App = {
  render() {
    return h("div", null, [h("div", null, String("123"))]);
  },
};
createApp(App).mount("#app");
```

```
else if (url.startsWith("/@modules/")) {
    // 这是一个node_module里的东西
    const prefix = path.resolve(
      __dirname,
      "node_modules",
      url.replace("/@modules/", "")
    );
    const module = require(prefix + "/package.json").module;
    const p = path.resolve(prefix, module);
    const ret = fs.readFileSync(p, "utf-8");
    ctx.type = "application/javascript";
    ctx.body = rewriteImport(ret);
  }
```

SFC组件支持

```
const compilerSfc = require("@vue/compiler-sfc"); // .vue
const compilerDom = require("@vue/compiler-dom"); // 模板

  else if (url.endsWith(".css")) {
    const p = path.resolve(__dirname, url.slice(1));
    const file = fs.readFileSync(p, "utf-8");
    const content = `
    const css = "${file.replace(/\n/g, "")}"
    let link = document.createElement('style')
    link.setAttribute('type', 'text/css')
    document.head.appendChild(link)
    link.innerHTML = css
    export default css
    `;
    ctx.type = "application/javascript";
    ctx.body = content;
  } else if (url.indexOf(".vue") > -1) {
    // vue单文件组件
    const p = path.resolve(__dirname, url.split("?")[0].slice(1));
    const { descriptor } = compilerSfc.parse(fs.readFileSync(p, "utf-8"));

    if (!query.type) {
      ctx.type = "application/javascript";
      // 借用vue自导的compile框架 解析单文件组件，其实相当于vue-loader做的事情
      ctx.body = `
  ${rewriteImport(
    descriptor.script.content.replace("export default ", "const __script = ")
  )}
  import { render as __render } from "${url}?type=template"
  __script.render = __render
```

```
  export default __script
      `;
    } else if (query.type === "template") {
      // 模板内容
      const template = descriptor.template;
      // 要在server端吧compiler做了
      const render = compilerDom.compile(template.content, { mode: "module" })
        .code;
      ctx.type = "application/javascript";

      ctx.body = rewriteImport(render);
    }
  }
```

三、容器化Docker技术

Docker概念

- 操作系统层面的虚拟化技术
- 隔离的进程独立于宿主和其它的隔离的进程 - 容器
- GO语言开发

特点

- 高效的利用系统资源
- 快速的启动时间
- 一致的运行环境
- 持续交付和部署
- 更轻松的迁移

对比传统虚拟机总结

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 `MB`	一般为 `GB`
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

三个核心概念

- 镜像
- 容器
- 仓库

1. 启动Nginx
    1. 拉取官方镜像

```
# 拉取官方镜像
docker pull nginx

# 查看
docker images nginx

# 启动镜像
mkdir www
echo 'hello docker!!' >> www/index.html

# 启动
# www目录里面放一个index.html
docker run -p 80:80 -v $PWD/dist:/usr/share/nginx/html -d nginx

# 查看进程
docker ps
docker ps -a // 查看全部

# 伪终端 ff6容器的uuid
# -t 选项让Docker分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上，
# -i 则让容器的标准输入保持打开
docker exec -it ff6 /bin/bash

# 停止
docker stop ff6

# 删除镜像
docker rm ff6
```

四、 持续集成 CI/CD 与 Github Action

- 自动发布
- 自动回归测试、代码检查

五、如何参见开源

> 跟我一起写Vue3版Element https://juejin.cn/post/6864462363039531022
>
> 如何参加开源项目-如何给Vue3.0提PR https://juejin.cn/post/6844904191744278542

六、Jest单元测试