Node.js 环境搭建与前端项目初始化
本章介绍

Node.js 环境搭建与初始化

- Node.js 环境配置与依赖包管理工具 Yarn 的安装与使用，讲解前端工程化开发思路。
- 学习 WebStrom 编辑器配置，常见前端开发插件推荐，提升 Vue.js 与 TS 开发效率。
- 学习 LLMOps 项目前端架构与基础框架选择，统一前端项目开发规范与目录约定。

项目的状态 / 样式 /Fetch 请求库的配置

- 介绍 TailwindCSS 及如何在项目中集成，利用 TailwindCSS 简化页面样式开发难度。
- 介绍 Vue.js 的状态管理库 Pinia 及使用技巧，如何在项目中使用 Pinia 在不同组件之间传递数据。
- 封装 JS 原生 Fetch 请求库，简化项目 API 接口请求难度。

Node.js 环境搭建及镜像加速

Node.js环境搭建及镜像加速

01. Node.js 环境搭建

Node.js 是 JavaScript 的运行时环境（Runtime Environment），能使 JavaScript 运行在服务端的运行环境。简单地讲，Node.js 可以使 JavaScript 像 Python/Java 一样在机器上运行，而不依赖浏览器。

Node.js 官网：https://nodejs.org/en。

通过官网安装下载的 Node.js 安装包进行安装，会自动将 Node.js 添加到系统的环境变量，无需手动设置，打开命令行执行：

```
node -v

# 输出
v20.14.0
```

在电脑上编写的 js 文件，可以通过 node 直接运行（类似 Python），例如：

```
// hello.js
console.log("Hello World!")
```

```
node hello.js

# 输出
Hello World!
```

或者使用 `node` 命令进入到命令行交互界面（类似 Python）：

```
node
```

02. npm 镜像加速配置

安装 Node.js 后，环境自带了 npm 包管理工具，通过 npm 可以便捷下载对应的第三方 JS/TS 包。

默认配置的下载源地址为https://registry.npmmirror.com，国内访问速度非常慢。

可以通过以下命令查看对应的 npm 配置信息（下载源）：

```
npm config list
```

输出如下：

```
; "builtin" config from D:\Env\nodejs\node_global\node_modules\npm\npmrc

; prefix = "C:\\Users\\Administrator\\AppData\\Roaming\\npm" ; overridden by user

; "user" config from C:\Users\Administrator\.npmrc

cache = "D:\\Env\\nodejs\\node_cache"
prefix = "D:\\Env\\nodejs\\node_global"
```

```
registry = "https://mirrors.cloud.tencent.com/npm/"

; node bin location = D:\Env\nodejs\node.exe
; node version = v20.9.0
; npm local prefix = C:\Users\Administrator
; npm version = 10.7.0
; cwd = C:\Users\Administrator
; HOME = C:\Users\Administrator
; Run `npm config ls -l` to show all defaults.
```

可以考虑将 npm 的镜像源设置为国内的镜像源，例如设置为腾讯提供的镜像源：

```
npm config set registry https://mirrors.cloud.tencent.com/npm/
```

校验镜像源是否设置成功：

```
npm config get registry
```

03. 依赖管理工具——Yarn

Node.js 环境中虽然自带了包管理工具——npm，但是使用起来缺陷也非常多：

1. 安装速度慢，npm 按队列安装，非并行安装。
2. npm 无法支持离线模型，对安装/缓存过的包，仍然需要从网络中下载（旧版本）。
3. npm 输出信息比较冗长，命令行里会不断地打印出所有被安装上的依赖，难以排查错误。
4. npm 必须通过 npm shrinkwrap 命令才可以生成锁定文件，并且版本管理特别容易混乱。
5. 在依赖包复杂的时候，npm 容易出现安装故障，导致依赖包安装失败。
6. 使用 npm 安装同一个项目时，安装时由于 package.json 文件中版本包的特性，安装无法保持一致性。
    - "5.0.3" 表示安装指定的 5.0.3 版本。
    - "~5.0.3" 表示安装的是 5.0.X 中的最新版本。
    - "^5.0.3" 表示安装 5.X.X 中的最新版本。

在这样的背景下，Yarn 就诞生了。

Yarn 是 Facebook、Goolge 等企业联合推出的一个新的 JS 包管理工具，正如官网所说的，Yarn 是为了弥补 npm 的一些缺陷而出现的，特别是新的项目拉下来要半天，删除 node_modules，重新 install 又需要半天。

Yarn 官网：https://yarnpkg.com/

尽管 Yarn 并不是 npm 包，但是可以通过 npm 来全局安装 Yarn，`-g` 命令代表全局，全局安装后，Yarn 会被安装到系统路径中：

```
npm install -g yarn
```

设置 Yarn 镜像源（腾讯镜像源）：

```
yarn config set registry https://mirrors.cloud.tencent.com/npm/
```

验证 Yarn 镜像源：

```
yarn config get registry
```

安装好 Yarn 之后，使用起来也非常简单，按照工作流任务的划分，有以下几个常见命令：

查看 Yarn 版本

```
yarn -v
```

初始化一个新项目

```
yarn init
```

添加依赖包

```
yarn add [package]
yarn add [package]@[version]
yarn add [package]@[tag]
```

将依赖项添加到不同依赖类别中

```
yarn add [package] --dev
yarn add [package] --peer
yarn add [package] --optional
```

升级依赖包

```
yarn upgrade [package]
yarn upgrade [package]@[version]
yarn upgrade [package]@[tag]
```

移除依赖包

```
yarn remove [package]
```

安装项目的所有依赖包

```
yarn
yarn install
```

项目前端架构与基础框架选择

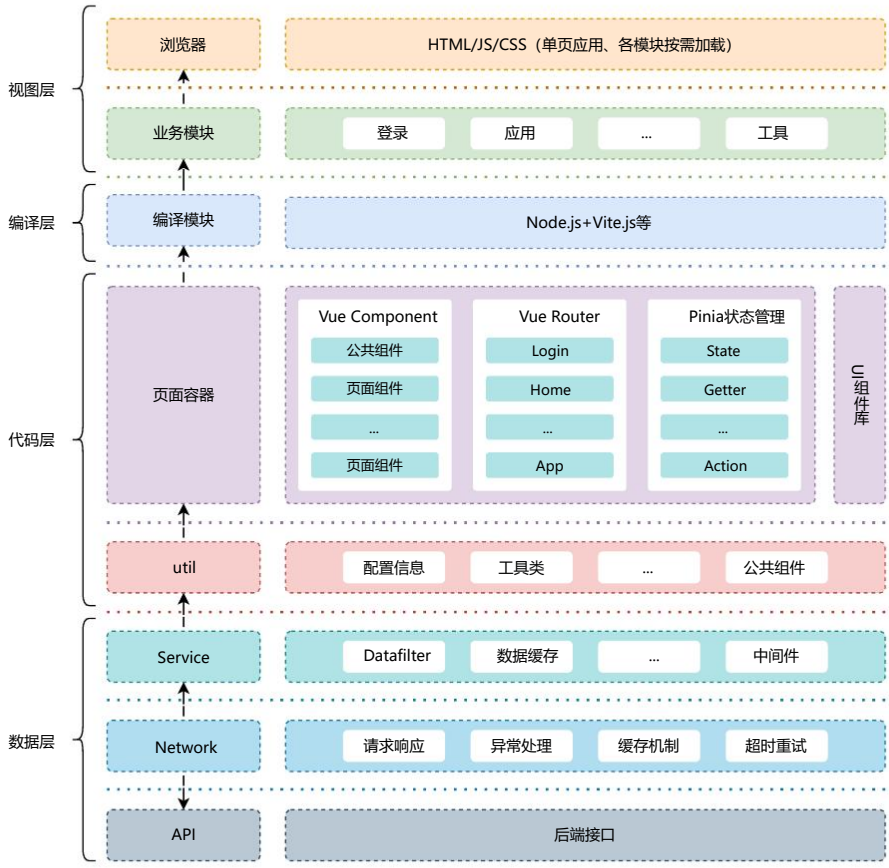
项目前端架构与基础框架选择

01. 项目前端架构

LLMOps 项目的前端部分包含了视图层、编译层、代码层和数据层 4 个部分。

1. 视图层：涵盖浏览器和业务模块，主要功能是将用户选择的 web 资源，通过解析网页源文件进行显示，并且不同的业务模块。
2. 编译层：编译层分析项目结构，根据入口文件找到 JavaScript 模块及其它浏览器不能直接运行的拓展语言（如TypeScript、SCSS等），通过 Vite 将源代码编译成适合浏览器使用的格式。
3. 代码层：在代码层，每个独立的可视或可交互区域都视为一个组件，并将所需的各种资源集中在组件目录下维护。Vue-router用于页面路径和组件的映射，vuex集中管理应用状态，UI组件库提高界面设计效率，util文件夹则管理全局工具。
4. 数据层：数据层包括Service模块，负责业务逻辑的独立性和重用性；Network模块，查看网络请求的内容；以及 Api 模块，通过 fetch 请求从后端接口获取所需数据。

项目技术架构图如下：



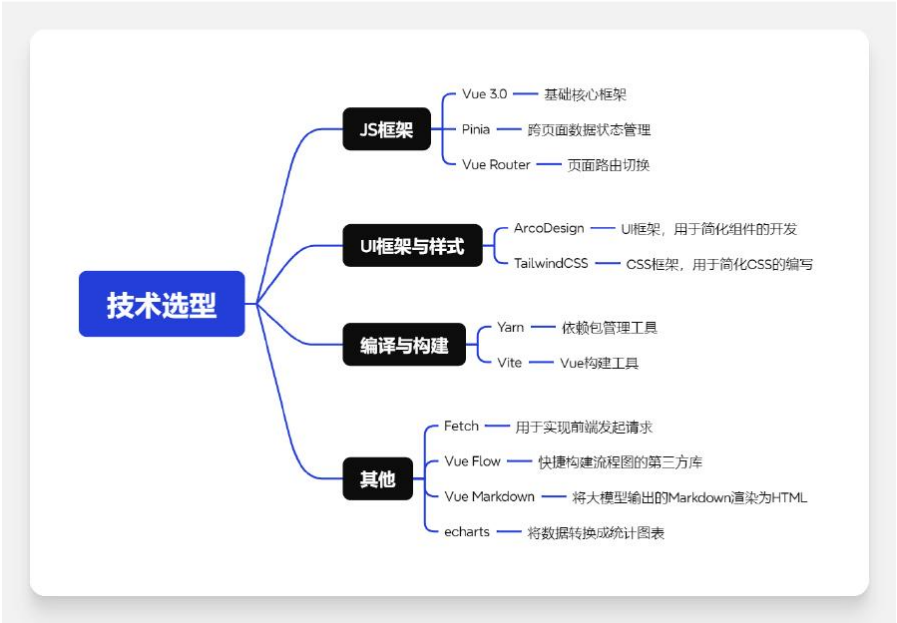
02. 项目技术选型

选定一个前端框架时，我们需要考虑的因素有：

- 框架是否能满足大部分应用的需求？如果不能，那么需要使用哪个框架？
- 框架是否有丰富的组件库？如果没有，我们的团队和组织是否有独立开发的能力？
- 框架的社区支持怎样？在遇到问题时能否快速方便地找到人解答？
- 框架的替换成本如何？假如我们的新项目将使用B框架，那么我们还需要额外学习什么内容？

目前市面上的三大前端框架 Angular、React、Vue，为什么选择 Vue 作为最基础的框架？

1. React：React 只是一个 View 层的框架，为了完成一个完整的应用，还需要使用路由库、执行单向流库、webAPI调用库、测试库、依赖管理、编译构建配置等，要搭建一个完整的 React 功能，需要做大量的额外工作，需要单独学习 JSX 模板语法。
2. Angular：一个大而全的框架，提供了开发应用所需的脚手架，包含测试、运行服务、打包等部分，但是上手成本高，框架限制高，对新手不友好，国内企业使用得比较少，并且于 2022 年已经停止维护。
3. Vue：对于没有 Angular 和 React 经验的团队来说，Vue 是一个非常好的选择。Vue 借鉴了 Angular 和 React 的一些思想，在其基础上开发了一套更易上手的框架。Vue 的开发者尤雨溪是中国人，框架本身提供了大量丰富的中文文档，这也为 Vue 的发展和使用带来巨大的优势。



前端项目搭建与开发规范

前端项目搭建与开发规范

01. 创建 Vue 项目与 Yarn 设置

1.2 创建 Vue 项目

Vue 官方提供了一个 `create-vue` 的脚手架，通过脚手架可以快速创建一个完整的 Vue 项目，涵盖：路由、状态管理、测试、代码格式化、构建配置等内容，而不需要手动去配置每一个部分。

Vue.js 官网：https://cn.vuejs.org/

创建命令也非常简单，运行 npm 或者 yarn 命令即可：

```
# npm创建vue项目
npm create vue@latest

# yarn创建vue项目
yarn create vue@latest
```

这个指令将会安装并执行 create-vue，它是 Vue 官方的项目脚手架工具。你将会看到一些诸如 TypeScript 和测试支持之类的可选功能提示：

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add an End-to-End Testing Solution? … No / Cypress / Nightwatch / Playwright
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

如果不确定是否要开启某个功能，你可以直接按下回车键选择 `No`。在项目被创建后，通过以下步骤安装依赖并启动开发服务器：

```
# yarn安装依赖以及启动开发服务器
cd <your-project-name>
yarn
yarn dev
```

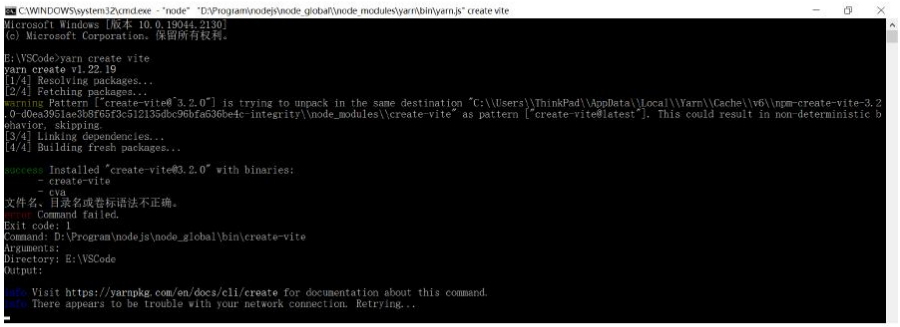
当准备将应用发布到生产环境时，可以运行以下命令：

```
# npm命令
npm run build

# yarn命令
yarn build
```

这个命令会在 `./dist` 文件夹中为应用构建一个生产环境的构建版本。

1.2 Yarn 设置

如果在使用 Yarn 创建 Vue 项目时，如果出现如下的报错：



出现这个错误的原因是 Yarn 的路径命令与其全局安装路径不在一个硬盘分区导致的。可以通过如下命令查看 Yarn 的命令路径和全局安装路径：

```
# 查看Yarn的命令路径
yarn global bin

# 查看Yarn的全局安装路径
yarn global dir
```

只需要将 Yarn 的全局安装路径和缓存路径设置到命令路径的同一个硬盘分区即可：

```
yarn config set global-folder "D:\Env\nodejs\yarn_global"
yarn config set cache-folder "D:\Env\nodejs\yarn_cache"
```

配置好命令后，将 `@latest` 去除运行即可，因为 Yarn 在 1.x.x 的版本下不支持 `@latest` 的写法。

```
yarn create vue
```

02. 项目目录约定

和后端项目一样，前端项目也有对应的目录规范约束，这里我们在 Vue.js 官方推荐的目录结构上进行衍生：

```
|---public   // 项目的公共资源文件夹
|---src   // 项目的代码文件夹
|   ├---assets // 资源文件夹: 图片/样式/图标
|   |   ├---images
|   |   ├---styles
|   ├---components   // 组件文件夹
|   |   ├---__tests__   // 测试文件夹
|   ├---config   // 项目配置文件夹
|   |   ├---index.ts
|   ├---hooks   // hooks文件夹
|   ├---router   // 路由文件夹
|   |   ├---index.ts
```

```
|   ├---services   // 项目服务文件夹
|   |   ├---apps.ts
|   |   ├---datasets.ts
|   ├---stores   // 状态管理文件夹
|   |   ├---account.ts
|   |   |---auth.ts
|   ├---utils   // 工具文件夹
|   |   ├---auth.ts
|   |   ├---request.ts
|   |   ├---storage.ts
|   ├---views   // 项目页面文件夹
|       ├---layouts
|           ├---DefaultLayout.vue
|           ├---BlankLayout.vue
|   |   ├---pages
|   |   |   ├---HomeView.vue
|   |   ├---auth
|   |   |   ├---LoginView.vue
|   |   ├---App.vue   // 应用主组件
|   |   ├---main.ts   // 应用js入口文件
|---.eslintrc.cjs   // 代码格式化配置
|---.gitignore   // git忽略配置
|---.prettierrc.json   // 代码美化配置
|---env.d.ts   // 项目TypeScript变量类型声明
|---index.html   // 应用html入口文件
|---package.json   // 依赖包管理
|---README.md   // 阅读说明
|---tsconfig.app.json   // ts配置
|---tsconfig.json
|---tsconfig.node.json
|---tsconfig.vitest.json
|---vite.config.ts   // vite编译配置
|---vitest.config.ts   // 测试框架配置
|---yarn.lock   // yarn锁
```

03. Vue.js 代码开发规范

规范的目的是为了提高团队协作效率，让团队当中其他人看你的代码可以一目了然，甚至一段时间后你再看某个时间写的代码也能很容易看清，越规范，越自由！

3.1 普通变量命名规范

- 命名方法： 驼峰命名法
- 命名规范：
    1. 命名必须是跟需求的内容相关的词，比如说我想申明一个变量，用来表示我的学校，那么我们可以这样定义 `const mySchool = "我的学校"；`
    2. 命名是复数的时候需要加 s ，比如说我想申明一个数组，表示很多人的名字，那么我们可以这样定义 `const names = []；`

3.2 常量

- 命名方法 : 全部大写
- 命名规范 : 使用大写字母和下划线来组合命名，下划线用以分割单词。

```
const MAX_COUNT = 10
const URL = 'https://www.imooc.com/'
```

3.3 组件命名规范

PascalCase（单词首字母大写命名）是最通用的声明约定

kebab-case（短横线分隔命名）是最通用的使用约定

- 组件名应该始终是多个单词的，根组件 App 除外
- 有意义的名词、简短、具有可读性
- 命名遵循 PascalCase 约定
  - 页面内部组件以组件模块名简写为开头，Item 为结尾，如（`StaffBenchToChargeItem`，`StaffBenchAppNotArrItem`）
- 使用遵循 kebab-case 约定
  - 在页面中使用组件需要前后闭合，并以短线分隔，如（`<abcd-date-picker></abcd-date-picker>`，`<abcd-table/>`）
- 导入及注册组件时，遵循 PascalCase 约定。
- 同时还需要注意：必须符合自定义元素规范，切勿使用保留字。

3.4 method 方法命名规范

- 驼峰式命名，统一使用动词或者动词+名词形式，见名知意。

```
//bad
go、nextPage、show、open、login

// good
jumpPage、openCarInfoDialog
```

3.5 views 下的文件命名

- 尽量是名词，且使用驼峰命名法
- 开头的单词就是所属模块名字（workbenchIndex、workbenchList、workbenchEdit）
- 名字至少两个单词（good: workbenchIndex）（bad:workbench）
- 页面使用 View 作为结尾，布局文件使用 Layout 作为结尾，其他组件

3.6 元素特性的顺序

原生属性放前面，指令放后面，如下所示：

```
- class
- id,ref
- name
```

- data-*
- src, for, type, href,value,max-length,max,min,pattern
- title, alt，placeholder
- aria-*, role
- required,readonly,disabled
- is
- v-for
- key
- v-if
- v-else-if
- v-else
- v-show
- v-cloak
- v-pre
- v-once
- v-model
- v-bind,:
- v-on,@
- v-html
- v-text

3.7 注释规范

1. 为公共组件使用添加说明；
2. 为组件中重要函数或者类添加说明；
3. 为复杂的业务逻辑处理添加说明；
4. 特殊情况的代码处理说明，对于代码中特殊用途的变量、存在临界值、函数中使用的 hack、使用了某种算法或思路等需要进行注释描述；
5. 多重 if 判断语句添加注释；
6. 注释块必须以 `/**（至少两个星号）开头**/`。
7. 单行注释使用 //。

3.8 编码规范

优秀的项目源码，即使是多人开发，看代码也如出一人之手。统一的编码规范，可使代码更易于阅读，易于理解，易于维护。尽量按照 ESLint 格式要求编写代码，写上完整的类型推断。

WebStorm 编辑器配置
提升 TS 代码开发效率

ArcoDesign 与 TailwindCSS
简化 UI 界面开发

ArcoDesign 与 TailwindCSS 简化 UI 界面开发

01. ArcoDesign UI 组件库

ArcoDesign 是字节跳动推出的 UI 组件库，拥有系统的设计规范和资源，依据此规范提供了覆盖 React、Vue 的原子组件。基于原子组件，Arco 提供了除风格配置平台、物料平台的定制化工具外还包括图标平台、品牌库、Arco Pro 最佳实践的资源平台。

安装：

```
# npm
npm install --save-dev @arco-design/web-vue

# yarn
yarn add --dev @arco-design/web-vue
```

在 `main.js` 中完整引入：

```
import { createApp } from 'vue'
import ArcoVue from '@arco-design/web-vue'
import App from './App.vue';
import '@arco-design/web-vue/dist/arco.css'

const app = createApp(App)
app.use(ArcoVue)
app.mount('#app')
```

在组件中即可使用 `<a-xxx></a-xxx>` 的方式来使用 ArcoDesign 设计好的组件，例如：

```
<template>
  <a-space>
    <a-button type="primary">Primary</a-button>
    <a-button>Secondary</a-button>
    <a-button type="dashed">Dashed</a-button>
    <a-button type="outline">Outline</a-button>
    <a-button type="text">Text</a-button>
  </a-space>
</template>
```

02. TailwindCSS 介绍与使用

2.1 TailwindCSS 介绍

TailwindCSS 是一个 CSS 框架，主要特点是实现了原子化 css，即 1 个 class 代表 1 个 css 属性，例如：

Class	css 属性	样式说明
`inline-block`	`display: inline-block;`	行内块元素

Class	css 属性	样式说明
p-2	padding: 0.5rem; /* 8px */	内边距为 8px
w-3	width: 0.75rem; /* 12px */	宽度为 12px
pr-px	padding-right: 1px;	右侧内边距为 1px
text-sky-400	color: rgb(56 189 248);	字体颜色为rgb(56, 189, 248)

不同 css 方案的解决场景对比：

```
<!-- 内联css -->
<div style="display: inline-block; background-color: rgb(14 165 233); padding:
14px">点击</div>

<!-- tailwindcss -->
<div class="inline-block bg-sky-500 p-4">点击</div>

<!-- 前端纯UI框架，比如 bootstrap -->
<div class="button">点击</div>

<!-- 前端组件库，比如 element-plus -->
<Button type="primary">点击</Button>
```

从上面的几种方案可以看到，不同 css 方案的颗粒度是逐步增大并内聚，自由度降低，`tailwindcss` 位于第二层。

2.2 TailwindCSS 安装配置

TailwindCSS 官网：https://tailwindcss.com/

安装与生成 `tailwind.config.js` 文件：

```
yarn add tailwindcss postcss autoprefix --dev
npx tailwindcss init -p
```

修改 `tailwind.config.js` 文件配置，添加编译的模板路径：

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{vue,js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

将 Tailwind 指令添加到 css 中：

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

在 vue 组件中使用 TailwindCSS：

```
<template>
  <h1 class="text-3xl font-bold underline">
    Hello world!
  </h1>
</template>
```

项目页面模板与路由配置
实现路由守卫功能

项目页面模板与路由配置，实现路由守卫功能

01. Vue Router 快速上手

1.1 Vue Router 简介

Vue-Router 是 Vue 官方的客户端路由解决方案。客户端路由的作用是在单页应用（SPA）中将浏览器的 URL 和用户看到的内容绑定起来。当用户在应用中浏览不同页面时，URL 会随之更新，但页面不需要从服务器重新加载。

Vue-Router 基于 Vue 的组件系统构建，可以通过配置路由来告诉 Vue Router 为每个 URL 路径显示哪个页面。

例如根组件 `App.vue`，代码如下：

```
<template>
  <h1>Hello App!</h1>
  <p>
    <strong>Current route path:</strong> {{ $route.fullPath }}
  </p>
  <nav>
    <RouterLink to="/">Go to Home</RouterLink>
    <RouterLink to="/about">Go to About</RouterLink>
  </nav>
  <main>
    <RouterView />
  </main>
</template>
```

在 `App.vue` 中使用了两个由 Vue Router 提供的组件 `RouterLink` 和 `RouterView`。

1. `RouterLink`：不同于常规的 `<a>` 标签，使用组件 `RouterLink` 来创建链接，这使得 Vue Router 能够在不重新加载页面的情况下改变 URL，处理 URL 的生成、编码和其他功能，本质上就是起到跳转页面的作用，其中参数 `to` 为需要跳转的页面。
2. `RouterView`：告知 Vue Router 你想要在哪里渲染当前 URL 路径对应的路由组件，它不一定要在 `App.vue` 中，你可以把它放在任何地方，但它需要在某处被导入，否则 Vue Router 不会渲染任何东西。

1.2 路由配置

页面想要使用 `RouterView` 和 `RouterLink` 必须配置对应的路由器，路由器实例通过 `createRouter` 进行配置，如下：

```
import { createMemoryHistory, createRouter } from 'vue-router'

import HomeView from './HomeView.vue'
import AboutView from './AboutView.vue'

const routes = [
  { path: '/', component: HomeView },
  { path: '/about', component: AboutView },
]
```

```
const router = createRouter({
  history: createMemoryHistory(),
  routes,
})
```

`routes` 定义了一组路由，将对应的 URL 路径映射到组件，其中 `component` 参数指定的组件就是先前在 `App.vue` 中被 `RouterView` 渲染的组件，这些组件通常被称为视图/布局，但本质上它们仍然也只是一个普通的组件。

除此之外，还可以给对应的 `路由映射关系` 进行命名，添加 `name` 属性即可。例如：

```
const routes = [
  { path: '/', name: 'home', component: HomeView },
  { path: '/about', name: 'about', component: AboutView },
]
```

`路由映射关系` 一旦命名，除了可以通过 URL 进行访问，也可以通过名称进行访问。

最后 `createRouter` 创建好路由器之后，需要被应用使用才可以起作用，这一步骤可以通过调用 `use()` 来完成。

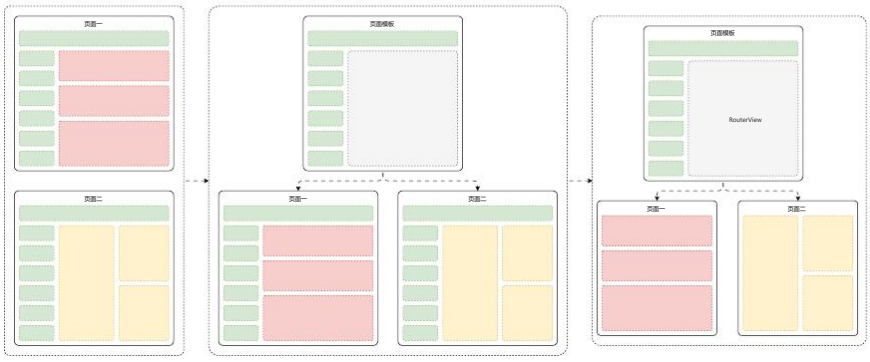
```
const app = createApp(App)
app.use(router)
app.mount('#app')
```

02. 页面嵌套布局

在 Vue Router 中，`RouterView` 不仅仅可以用于 `App.vue` 组件中，还可以在任何组件中使用，被使用的组件，在进行路由配置时，可以添加 `childrens` 属性，用于配置该组件需要渲染的子组件，并且子组件的渲染位置就是 `RouterView`，从而实现页面嵌套布局。

页面的嵌套布局可以减少页面 UI 组件的重复性开发，提升代码的复用性和可维护性，例如下方就是一个页面嵌套布局的写法。

1. `页面一` 与 `页面二` 拥有大量的共同元素，只有内部部分区域不一样。
2. 可以考虑从两个页面中提取出一个公共模板，也就是 `页面模板`。
3. 然后创建 `页面模板`，并配置 `RouterView` 和 `childrens`，让两个页面作为 `页面模板` 的子组件，从而实现嵌套。

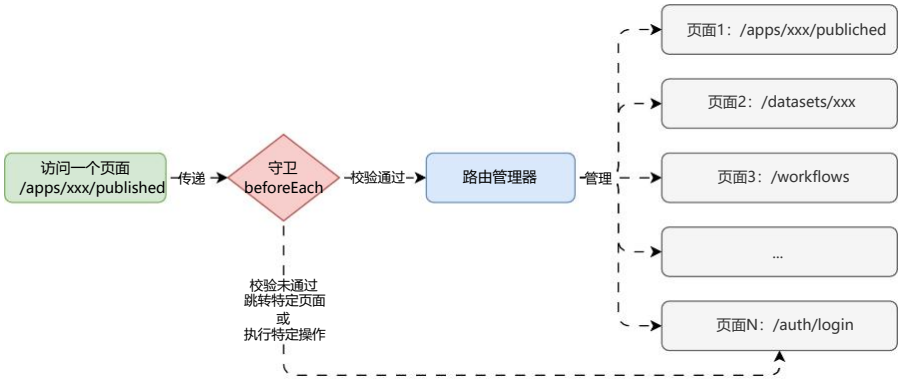


03. 路由守卫功能

Vue-Router 提供的路由守卫主要用来通过跳转或取消的方式来守卫路由。简单来说，默认情况下，用户访问一个页面，请求会直接传递给路由管理器，路由管理器找到对应的页面组件之后返回给浏览器进行渲染。

如果添加了 `路由守卫功能(全局路由守卫)`，则会在路由器前添加一个校验逻辑，用于校验当前场景是否可以访问后续的页面，如果可以则放行，页面可以正常访问；如果不满足条件，则执行特定的操作，或者跳转到特定的页面等。

路由守卫的运行流程如下：



使用示例：

```
router.beforeEach(async (to, from) => {
  // 检查账号是否登录
  if (!auth.isLogin() && !['auth-login', 'auth-authorize'].includes(to.name as
string)) {
    return { path: '/auth/login' }
  }
})
```

Pinia 实现多页面共享数据状态

Pinia实现多页面共享数据状态

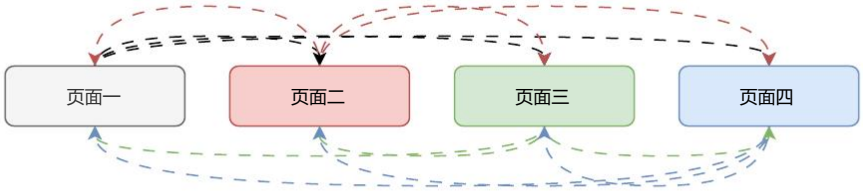
01. Vue 状态管理难点

在 Vue 中，不同的页面之间在某些场景下会共享同一份数据，例如：账号登录信息、页面的黑暗模式、授权凭证等，通过页面与页面之间的传参进行数据共享，数据状态难以维护，并且开发效率慢，于是出现了 `状态管理库` 的概念。

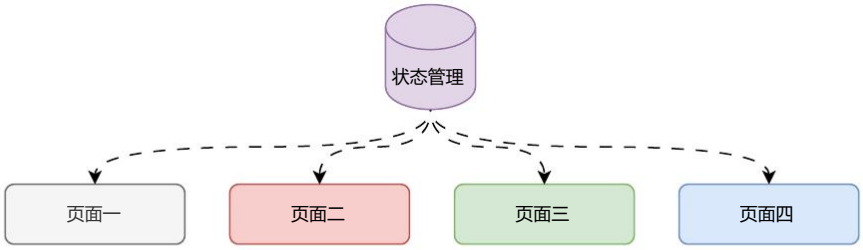
`状态管理库` 统一维护整个项目共享的数据，在每个页面之间均可以任意使用，共享数据的更新，降低了页面组件之间传参的难度。

使用 `状态管理库` 和不使用时，页面的数据传递流程图对比：

使用状态管理器前的数据流向：



使用状态管理器后的数据流向：



02. Pinia 快速上手

Pinia 是 Vue 的存储库，它允许在项目中跨组件/页面共享状态。简单来说，可以将 Pinia 看成是 Vue 的临时内存数据库，在 Pinia 内存储的数据，可以被 Vue 的所有页面/组件访问，并且同步更新。

`pinia` 的安装：

```
yarn add pinia
# 或者使用 npm
npm install pinia
```

`pinia` 的配置：

```
import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'

const pinia = createPinia()
const app = createApp(App)

app.use(pinia)
app.mount('#app')
```

Pinia 使用有 3 大要素：数据、计算属性、函数。

1. `数据`：或者说是状态，是 Pinia 存储的数据结构。
2. `计算属性`：或者说是 `getter`，是基于数据计算后得到的结果，只读属性。
3. `函数`：或者说是动作，用于处理数据，涵盖数据的增、删、改等操作。

数据状态存储定义示例：

```
import { ref, computed } from 'vue'
import { defineStore } from 'pinia'

export const useCounterStore = defineStore('counter', () => {
  // 1.创建存储的数据
  const count = ref(0)

  // 2.创建计算属性
  const doubleCount = computed(() => count.value * 2)

  // 3.创建数据处理动作
  function increment() {
    count.value++
 }

  return { count, doubleCount, increment }
})
```

数据状态存储使用示例：

```
<script setup lang="ts">
import { useCounterStore } from '@/stores/counter'

const counterStore = useCounterStore()
</script>

<template>
  <p>当前的值为:{{counterStore.count}}</p>
  <p>两倍计算的值为:{{counterStore.doubleCount}}</p>
  <button @click="counterStore.increment">数值+1</button>
</template>
```

前端接口请求 Fetch 方法封装

前端接口请求Fetch方法封装

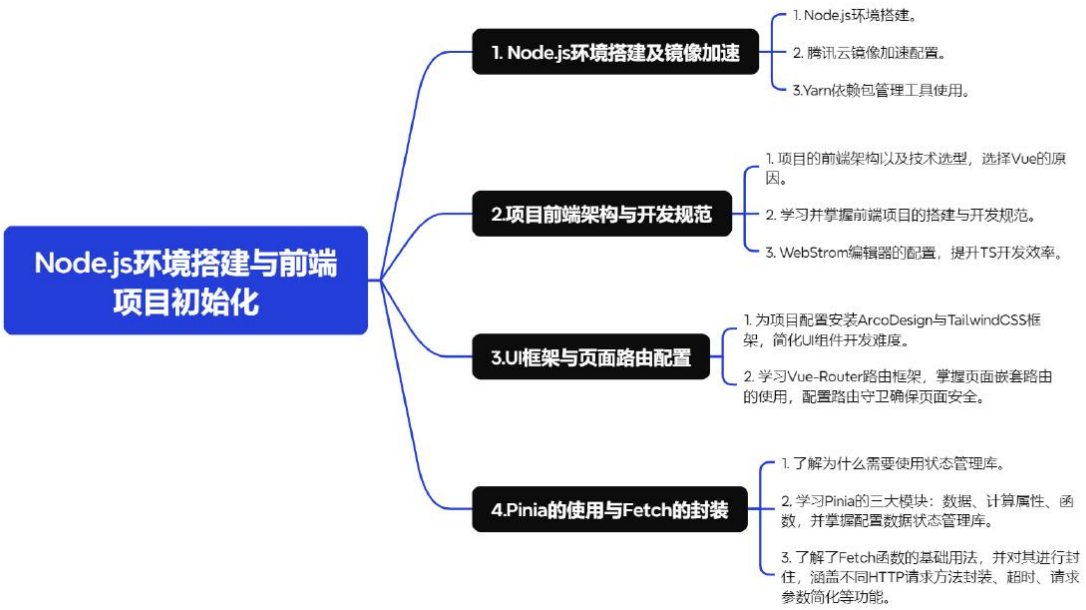
01. Fetch 基础用法

MDN 官网 fetch 函数文档: https://developer.mozilla.org/zh-CN/docs/Web/API/Fetch_API/Using_Fetch

02. Fetch 封装思路

```
import { apiPrefix } from '@/config'

// 超时时间100s
const TIME_OUT = 100000

// Fetch请求参数类型
type FetchOptionType = Omit<RequestInit, 'body'> & {
  params?: Record<string, any>
  body?: BodyInit | Record<string, any> | null
}

// Fetch请求基础配置
const baseFetchOptions = {
  method: 'GET',
  mode: 'cors',
  credentials: 'include', // 每次都携带cookie、Http基础验证信息
  headers: new Headers({
    'Content-Type': 'application/json',
  }),
  redirect: 'follow',
}
// 封装基础fetch函数
const baseFetch = <T>(url: string, fetchOptions: FetchOptionType): Promise<T> =>
{
  // 1.获取传递的所有配置
  const options: typeof baseFetchOptions & FetchOptionType = Object.assign(
    {},
    baseFetchOptions,
    fetchOptions,
  )

  // 2.计算实际请求发起的URL地址
  let urlWithPrefix = `${apiPrefix}${url.startsWith('/') ? url : `/${url}`}`

  // 3.解构请求的method、params、body
  const { method, params, body } = options

  // 4.如果方法为GET并传递了参数，则处理参数信息
  if (method === 'GET' && params) {
    const paramsArray: string[] = []
    Object.keys(params).forEach((key) => {
      paramsArray.push(`${key}=${encodeURIComponent(params[key])}`)
    })
    if (urlWithPrefix.search(/\?/) === -1) {
```

```
        urlWithPrefix += `?${paramsArray.join('&')}`
      } else {
        urlWithPrefix += `&${paramsArray.join('&')}`
      }

      // 5.删除params参数
      delete options.params
    }

    // 6.处理传递的post参数
    if (body) options.body = JSON.stringify(body)

    // 7.构建超时处理
    return Promise.race([
      // 超时Promise
      new Promise((reject) => {
        setTimeout(() => {
          reject('请求超时')
        }, TIME_OUT)
      }),
      // 正常接口响应
      new Promise((resolve, reject) => {
        globalThis
          .fetch(urlWithPrefix, options as RequestInit)
          .then((res) => {
            resolve(res.json())
          })
          .catch((err) => {
            reject(err)
          })
      }),
    ]) as Promise<T>
}

// 封装基础的fetch请求
export const request = <T>(url: string, options = {}) => {
  return baseFetch<T>(url, options)
}

// 封装基础的get请求
export const get = <T>(url: string, options = {}) => {
  return request<T>(url, Object.assign({}, options, { method: 'GET' }))
}

// 封装基础的post请求
export const post = <T>(url: string, options = {}) => {
  return request<T>(url, Object.assign({}, options, { method: 'POST' }))
}
```


Node.js环境搭建与前端项目初始化
1. Node.js环境搭建及镜像加速
1. Node.js环境搭建。
2. 腾讯云镜像加速配置。
3.Yarn依赖包管理工具使用。
2.项目前端架构与开发规范
1. 项目的前端架构以及技术选型，选择Vue的原因。
2. 学习并掌握前端项目的搭建与开发规范。
3. WebStrom编辑器的配置，提升TS开发效率。
3.UI框架与页面路由配置
1. 为项目配置安装ArcoDesign与TailwindCSS框架，简化UI组件开发难度。
2. 学习Vue-Router路由框架，掌握页面嵌套路由的使用，配置路由守卫确保页面安全。
4.Pinia的使用与Fetch的封装
1. 了解为什么需要使用状态管理库。
2. 学习Pinia的三大模块：数据、计算属性、函数，并掌握配置数据状态管理库。
3. 了解了Fetch函数的基础用法，并对其进行封住，涵盖不同HTTP请求方法封装、超时、请求参数简化等功能。