Vue3 設置

安裝 Vue 3 並創建一個新的 Vue 3 項目是一個相對簡單的過程，可以通過幾個簡單的步驟完成。下面將指導您如何使用 Vue CLI（命令行界面）來完成這一過程。

Vue 3 設置

安裝 Node.js

首先，確保您已經安裝了 Node.js。Vue 3 需要 Node.js 的支持。您可以訪問 Node.js 官網 (https://nodejs.org/) 下載並安裝適合您操作系統的版本。

安裝 Vue CLI

安裝了 Node.js 之後，您就可以使用 npm（Node.js 的包管理器）來安裝 Vue CLI 了。打開您的終端機或命令提示符，執行以下命令：

```
npm install -g @vue/cli
```

這條命令會全局安裝 Vue CLI，讓您可以在任何地方使用它來創建 Vue 項目。

創建一個新的 Vue 3 項目

安裝 Vue CLI 之後，就可以創建一個新的 Vue 3 項目了。在終端機或命令提示符中，運行以下命令：

```
vue create my-vue3-project
```

這條命令會提示您選擇一個預設的配置或手動配置您的項目。對於大多數初學者來說，選擇預設配置即可。這將自動為您的新項目選擇 Vue 3。

如果您想手動配置並明確選擇 Vue 3，請選擇手動配置。在配置過程中，當問及您想使用哪個版本的 Vue 時，請選擇 Vue 3。

項目結構

創建完成後，您將得到一個基本的項目結構，大致如下：

- `node_modules/`：存放項目依賴的資料夾。
- `public/`：存放靜態資源如 `index.html`。
- `src/`：存放您的 Vue 源代碼，包括組件、圖片、樣式等。
    - `assets/`：存放圖片等資源文件。
    - `components/`：存放 Vue 組件。
    - `App.vue`：主組件文件。
    - `main.js`：Vue 應用的入口文件。
- `.gitignore`：指定 git 忽略的文件。
- `babel.config.js`：Babel 的配置文件。
- `package.json`：項目的描述文件，包含依賴、腳本等信息。
- `README.md`：項目的說明文件。

運行您的 Vue 3 項目

進入項目目錄，並運行開發伺服器：

```
cd my-vue3-project
npm run serve
```

這將啟動一個開發伺服器，通常您可以通過訪問 `http://localhost:8080` 在瀏覽器中預覽您的 Vue 應用。

這就是安裝 Vue 3 並創建一個新的 Vue 3 項目的全部過程。現在，您可以開始開發您的 Vue 應用了！

Yarn

Yarn 是一個快速、可靠、安全的 JavaScript 包管理工具，由 Facebook 開發並開源。它旨在解決客戶端 JavaScript 包的安裝、配置、更新以及依賴管理等問題，並致力於提高效率和保證包管理的一致性。Yarn 通過緩存每一個下載過的包，來提高安裝過程的速度。此外，它還可以並行處理依賴包的安裝，大幅提升安裝效率。

Yarn 的配置文件

Yarn 主要使用以下幾種配置文件：

- `package.json`：這是 Node.js 和 Yarn 的標準配置文件，用於指定項目的元數據、依賴包等信息。
- `yarn.lock`：當安裝依賴時，Yarn 會生成一個 `yarn.lock` 文件。這個文件鎖定了依賴包的版本，確保每次安裝時都能獲得相同版本的依賴，從而避免了在不同環境下因版本不一致而導致的問題。
- `.yarnrc` 或 `.yarnrc.yml`：這些是 Yarn 的配置文件，可以用來配置 Yarn 的行為，如設定代理、設定註冊表地址等。

如何開始使用 Yarn

1. **安裝 Yarn**

   首先，你需要安裝 Yarn。如果你已經安裝了 Node.js，可以通過 npm 來安裝 Yarn：

   ```
   npm install -g yarn
   ```

   或者，你可以通過其他方法安裝 Yarn，具體可參考 Yarn 官網 (https://yarnpkg.com/)上的安裝指南。

2. **初始化新項目**

   在項目目錄下運行以下命令來初始化一個新的 Yarn 項目。這會創建一個基本的 `package.json` 文件：

   ```
   yarn init
   ```

3. **添加依賴包**

   使用 Yarn 添加一個依賴包非常簡單，只需運行：

   ```
   yarn add [package_name]
   ```

   例如，要添加 Vue.js 為項目依賴：

   ```
   yarn add vue
   ```

4. **安裝所有依賴**

   如果你是在已有項目的基礎上工作，並且項目中包含 `package.json` 文件，只需運行以下命令來安裝所有依賴：

   ```
   yarn
   ```

5. **運行腳本**

   在 `package.json` 文件中定義的腳本可以通過 Yarn 來執行。例如，如果你有一個啟動開發服務器的腳本名為 "start"：

   ```
   yarn run start
   ```

範例：`package.json` 和 `yarn.lock`

假設你正在開發一個使用 Vue.js 的前端項目，你的 `package.json` 可能看起來像這樣：

```
{
  "name": "my-vue-app",
  "version": "1.0.0",
  "description": "A Vue.js project",
  "main": "index.js",
  "dependencies": {
    "vue": "^2.6.12"
  },
  "devDependencies": {},
  "scripts": {
    "start": "vue-cli-service serve"
  },
  "author": "",
  "license": "ISC"
}
```

當你運行 `yarn add vue` 後，Yarn 會在你的項目中創建一個 `yarn.lock` 文件，這個文件會記錄下安裝的 Vue.js 版本以及其所有依賴的確切版本，以保證項目的一致性。

`yarn.lock` 文件看起來可能會像這樣：

```
# THIS IS AN AUTOGENERATED FILE. DO NOT EDIT THIS FILE DIRECTLY.
# yarn lockfile v1


vue@^2.6.12:
  version "2.6.12"
  resolved "https://registry.yarnpkg.com/vue/-/vue-2.6.12.tgz#..."
  integrity sha512-...
  dependencies:
    compiler-sfc "^3.0.0"
```

這個文件中記錄了 Vue.js 的具體版本是 2.6.12，並且還記錄了它的依賴以及這些依賴的版本。這樣，無論是你自己還是你的團隊成員，在任何時候使用 `yarn` 命令安裝依賴時，都會得到完全相同版本的依賴，這對於確保開發和生產環境的一致性至關重要。

總結

Yarn 作為一個現代的包管理工具，它提供了比 npm 更快的依賴安裝速度、更嚴格的版本控制以及更好的跨平台支持。通過使用 `yarn.lock` 文件，Yarn 確保了所有開發者和 CI/CD 環境中的依賴版本完全一致，從而減少了「在我機器上可以運行，但是在你機器上就不行」這類問題的發生。

開始使用 Yarn，只需幾個簡單的步驟：安裝 Yarn、初始化新項目、添加依賴包、安裝所有依賴，以及運行項目中定義的腳本。這樣，你就可以享受到 Yarn 帶來的便捷和效率，為你的 JavaScript 或 TypeScript 項目帶來更好的開發體驗。

Babel

Babel 是一個廣泛使用的 JavaScript 轉譯器，它允許開發者使用最新的 JavaScript 語言特性，而不需要等待這些特性在所有的瀏覽器或環境中都被支持。簡單來說，Babel 可以將使用最新語法編寫的 JavaScript 代碼轉換成向後兼容的版本，這樣就能在當前的瀏覽器或其他環境中運行，無需擔心兼容性問題。

Babel 的配置文件

Babel 的配置文件是一個名為 `.babelrc` 或 `babel.config.json` 的文件，位於項目的根目錄下。這個配置文件允許開發者細化設置轉譯過程中的各種選項。

配置文件可以包括：

- **presets**：預設的一組插件，用於支持特定的語言特性。例如，`@babel/preset-env` 可以根據目標環境自動確定需要轉換的語言特性。
- **plugins**：插件可以用來支持更多的語言特性或進行代碼轉換的定制。

配置內容範例

下面是一個基本的 `.babelrc` 配置範例：

```
{
  "presets": ["@babel/preset-env"],
  "plugins": []
}
```

這個配置使用了 `@babel/preset-env` 預設，它根據你的目標環境自動選擇需要轉譯的 JavaScript 新特性。這個配置沒有指定任何插件。

使用範例

假設你有一段使用了 ES6 箭頭函數和模板字符串的代碼：

```
const greet = (name) => `Hello, ${name}!`;
```

如果你的目標環境不支持箭頭函數或模板字符串，Babel 可以將其轉譯成舊版 JavaScript：

```
var greet = function(name) {
  return "Hello, " + name + "!";
};
```

結論

Babel 是一個強大的工具，可以幫助開發者使用最新的 JavaScript 語言特性，同時確保代碼的廣泛兼容性。通過適當配置 `.babelrc` 或 `babel.config.json`，你可以細化控制轉譯過程，以符合你的開發需求。

```
npm list @babel/core
npm list @babel/cli
```

```
npm update @babel/core
npm update @babel/cli
```

ESLint

ESLint 是一個開源的 JavaScript 靜態代碼檢查工具，目的是幫助開發者找出代碼中的問題，保持代碼質量和風格的一致性。它可以識別出許多類型的錯誤，從語法錯誤到潛在的邏輯錯誤，並且可以根據開發者的配置來強制實施一套特定的編碼風格和規則。

ESLint 的配置文件

ESLint 的配置文件可以是 `.eslintrc.js`、`.eslintrc.json`、`.eslintrc.yaml`、`.eslintrc.yml`，或者直接在 `package.json` 文件中的 `eslintConfig` 屬性下配置。這些配置文件讓你可以細化設置 ESLint 的規則、插件、解析器選項等。

配置內容範例

下面是一個基本的 `.eslintrc.json` 配置範例：

```
{
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": "eslint:recommended",
  "parserOptions": {
    "ecmaVersion": 12,
    "sourceType": "module"
  },
  "rules": {
    "indent": ["error", 2],
    "linebreak-style": ["error", "unix"],
    "quotes": ["error", "single"],
    "semi": ["error", "always"]
  }
}
```

在這個配置中：

- `env` 指定了代碼運行的環境，這裡包括瀏覽器環境和 ES2021 特性。
- `extends` 使用了 `eslint:recommended`，這是一組 ESLint 官方推薦的規則。
- `parserOptions` 設定了 ECMAScript 版本和模塊類型，允許使用 ES2021 特性和 ECMAScript 模塊。
- `rules` 定義了具體的規則配置，例如縮進使用 2 個空格、換行風格為 Unix 風格、使用單引號和語句結尾需要分號。

使用範例

假設你有以下的 JavaScript 代碼：

```
function sum(a, b) {
  return a + b;
}
```

如果你的 ESLint 配置要求使用箭頭函數和單引號，那麼 ESLint 可能會報告一個警告或錯誤，指出應該使用箭頭函數並更改雙引號為單引號。

根據上面的規則，修改後的代碼可能如下：

```
const sum = (a, b) => {
  return a + b;
};
```

結論

ESLint 是一個非常有用的工具，對於提高代碼質量、發現潛在錯誤和保持團隊間代碼風格的一致性非常有幫助。通過配置文件，開發者可以自定義規則來符合他們的編碼標準和偏好。

ESLint

在 Vue 專案中使用 ESLint，可以幫助你維持代碼風格的一致性並避免常見錯誤。以下是如何在 Vue 專案中開始使用 ESLint 的步驟：

步驟 1：安裝 ESLint

首先，你需要確保你的開發環境已經安裝了 Node.js。然後，在你的 Vue 專案根目錄下，通過 npm 或 yarn 安裝 ESLint。打開終端（命令行工具）並運行以下命令之一：

```
# 使用 npm
npm install eslint --save-dev

# 使用 yarn
yarn add eslint --dev
```

步驟 2：初始化 ESLint

安裝完 ESLint 後，你需要對其進行初始化。這可以通過運行 ESLint 的初始化命令來完成，它會創建一個配置文件 `.eslintrc.js`，並讓你選擇一些基本配置。

```
npx eslint --init
```

過程中，它會問你幾個問題，比如你的代碼是否運行在瀏覽器或 Node.js 環境、是否使用了 ES6、是否使用了某個框架（如 Vue）等。根據你的 Vue 專案特性選擇相應的選項。

步驟 3：安裝 Vue 插件（可選）

為了更好地支持 Vue 檔案的 linting，你可能需要安裝一個 ESLint 插件專門為 Vue 準備。這個插件叫做 `eslint-plugin-vue`。

```
# 使用 npm
npm install eslint-plugin-vue --save-dev

# 使用 yarn
yarn add eslint-plugin-vue --dev
```

安裝完畢後，打開 `.eslintrc.js` 配置文件，確保 `eslint-plugin-vue` 被正確配置：

```
module.exports = {
  ...
  extends: [
    // 添加 Vue 的推薦規則
    'plugin:vue/vue3-recommended',
    // 或者如果你使用的是 Vue 2
    // 'plugin:vue/recommended'
  ],
  ...
};
```

步驟 4：在你的專案中使用 ESLint

你現在可以開始使用 ESLint 檢查你的 Vue 專案中的代碼了。可以在終端中運行 ESLint，檢查特定文件或整個專案：

```
# 檢查特定文件
npx eslint path/to/YourComponent.vue

# 檢查整個專案中的文件
npx eslint .
```

步驟 5：整合到開發流程中（可選）

你還可以將 ESLint 整合到你的開發流程中，例如，配置 package.json 的腳本，使其成為你的構建過程的一部分，或者在提交代碼前自動執行 lint 檢查。

```
"scripts": {
  "lint": "eslint --ext .js,.vue src"
}
```

這樣，你就可以通過運行 `npm run lint` 或 `yarn lint` 來檢查你的代碼。

通過這些步驟，你可以在你的 Vue 專案中成功地開始使用 ESLint，從而幫助你提高代碼質量和一致性。