Vue预习课

必备知识——Vue Cli

快速原型开发

你可以使用 `vue serve` 和 `vue build` 命令对单个 `*.vue` 文件进行快速原型开发。

安装`@vue/cli-service-global`扩展

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

准备一个内容原型

vue serve

启动一个服务并运行原型

```
vue serve Hello.vue
```

创建项目

vue create

创建一个vue项目

```
vue create my-vue-test
```

 CLI 预览

vue ui

图形化项目管理

```
vue ui
```

范例：移植之前创建的cart内容到当前项目中来

插件

Vue CLI 使用了一套基于插件的架构。插件可以修改 webpack 的内部配置，也可以向 `vue-cli-service` 注入命令。在项目创建的过程中，绝大部分列出的特性都是通过插件来实现的。

在现有的项目中安装插件

如果你想在一个已经被创建好的项目中安装一个插件，可以使用 `vue add` 命令。

```
vue add router
```

开发

处理资源路径

当你在 JavaScript、CSS 或 `*.vue` 文件中使用相对路径 (必须以 `.` 开头) 引用一个静态资源时，该资源将被webpack处理。

转换规则

- 如果 URL 是一个绝对路径 (例如 `/images/foo.png`)，它将会被保留不变。

```
<img alt="Vue logo" src="/assets/logo.png">
<img alt="Vue logo" src="http://image.xx.com/logo.png">
```

- 如果 URL 以 `.` 开头会作为一个相对模块请求被解释并基于文件系统相对路径。

```
<img alt="Vue logo" src="./assets/logo.png">
```

- 如果 URL 以 `~` 开头会作为一个模块请求被解析。这意味着你甚至可以引用 Node 模块中的资源：

```
<img src="~some-npm-package/foo.png">
```

- 如果 URL 以 `@` 开头会作为一个模块请求被解析。Vue CLI 默认会设置一个指向 `src` 的别名 `@`。

```
import Hello from '@/components/Hello.vue'
```

何时使用 `public` 文件夹

通过 webpack 的处理并获得如下好处：

- 脚本和样式表会被压缩且打包在一起，从而避免额外的网络请求。
- 文件丢失会直接在编译时报错，而不是到了用户端才产生 404 错误。
- 最终生成的文件名包含了内容哈希，因此你不必担心浏览器会缓存它们的老版本。

如下情况考虑使用public文件夹

- 你需要在构建输出中指定一个固定的文件名字。
- 你有上千个图片，需要动态引用它们的路径。
- 有些库可能和 webpack 不兼容，除了将其用一个独立的 `<script>` 标签引入没有别的选择。

使用public文件夹的注意事项

- 如果你的应用没有部署在域名的根部，那么你需要为你的 URL 配置 publicPath 前缀

```
// vue.config.js
module.exports = {
  publicPath: process.env.NODE_ENV === 'production'
    ? '/cart/'
    : '/'
}
```

- 在 `public/index.html` 等通过 `html-webpack-plugin` 用作模板的 HTML 文件中，你需要通过 `<%= BASE_URL %>` 设置链接前缀：

```
<link rel="icon" href="<%= BASE_URL %>favicon.ico">
```

- 在模板中，先向组件传入BASE_URL：

```
data () {
  return {
    publicPath: process.env.BASE_URL
  }
}
```

然后：

```
  <img :src="`${publicPath}my-image.png`">
```

CSS相关

使用预处理器

如果创建项目时没有选择需要的预处理器（Sass/Less/Stylus），则需手动安装相应loader

```
# Sass
npm install -D sass-loader node-sass

# Less
npm install -D less-loader less

# Stylus
npm install -D stylus-loader stylus
```

范例：App.vue修改为Sass

```
<style scoped lang="scss">
    $color: #42b983;
    a {
        color: $color;
    }
</style>
```

自动化导入样式

自动化导入样式文件 (用于颜色、变量、mixin等)，可以使用 style-resources-loader。

```
npm i -D style-resources-loader
```

配置

```
const path = require('path')

function addStyleResource(rule) {
    rule.use('style-resource')
        .loader('style-resources-loader')
        .options({
        patterns: [
            path.resolve(__dirname, './src/styles/imports.scss'),
        ],
    })
```

```
}

module.exports = {
    chainWebpack: config => {
        const types = ['vue-modules', 'vue', 'normal-modules', 'normal']
        types.forEach(type =>
addStyleResource(config.module.rule('scss').oneOf(type)))
    },
}
```

Scoped CSS

当 `<style>` 标签有 `scoped` 属性时，它的 CSS 只作用于当前组件中的元素。

```
<style scoped>
.red {
  color: red;
}
</style>
```

> 其原理是通过使用 PostCSS 来实现以下转换：
>
> ```
> <template>
> <div class="red" data-v-f3f3eg9>hi
> </template>
>
> <style>
> .red[data-v-f3f3eg9] {
> color: red;
> }
> </style>
> ```

混用本地和全局

```
<style>
/* 全局样式 */
</style>

<style scoped>
/* 本地样式 */
</style>
```

深度作用选择器：使用 `>>>` 操作符可以使 `scoped` 样式中的一个选择器能够作用得“更深”，例如影响子组件

```
<style scoped>
#app >>> a {
  color: red
}
</style>
```

> Sass 之类的预处理器无法正确解析 `>>>`。这种情况下你可以使用 `/deep/` 或 `::v-deep` 操作符取而代之
>
> ```
> <style scoped lang="scss">
> #app {
>   /deep/ a {
>     color: rgb(196, 50, 140)
>   }
>   ::v-deep a {
>     color: rgb(196, 50, 140)
>   }
> }
> </style>
> ```

CSS Module

CSS Modules 是一个流行的，用于模块化和组合 CSS 的系统。`vue-loader` 提供了与 CSS Modules 的一流集成，可以作为模拟 scoped CSS 的替代方案。

添加module

```
<style module lang="scss">
.red {
  color: #f00;
}
.bold {
  font-weight: bold;
}
</style>
```

模板中通过$style.xx访问

```
<a :class="$style.red">awesome-vue</a>
<a :class="{[$style.red]:isRed}">awesome-vue</a>
<a :class="[$style.red, $style.bold]">awesome-vue</a>
```

JS中访问

```
<script>
export default {
  created () {
    // -> "red_1VyoJ-uZ"
    // 一个基于文件名和类名生成的标识符
    console.log(this.$style.red)
  }
}
</script>
```

数据访问相关

数据模拟

使用开发服务器配置before选项，可以编写接口，提供模拟数据。

```
devServer:{
    before(app) {
        app.get('/api/courses', (req, res) => {
            res.json([{ name: 'web全栈', price: 8999 }, { name: 'web高级', price: 8999 }])
        })
    }
}
```

调用

```
import axios from 'axios'

export function getCourses() {
  return axios.get('/api/courses').then(res => res.data)
}
```

代理

设置开发服务器代理选项可以有效避免调用接口时出现的跨域问题。

```
devServer: {
    proxy: 'http://localhost:3000'
}
```

测试接口

```
// 需要安装express：npm i express
const express = require('express')
const app = express()

app.get('/api/courses', (req, res) => {
  res.json([{ name: 'web全栈', price: 8999 }, { name: 'web高级', price: 8999 }])
})

app.listen(3000)
```