Vue DAY02

使用 vue 开发 web 应用的编程方式

传统的 DOM 操作或 JQuery 的开发模式，如果需要更新 DOM，例如：

```
<div id="info">欢迎：<span id="name">张三</span></div>
```

```
$('#info').text('欢迎：亮亮')
$('#name').text('亮亮')
```

传统的写法，讲究的是通过选择器等方式找到页面中符合条件的 DOM 元素，然后修改它的属性。

vue的写法：

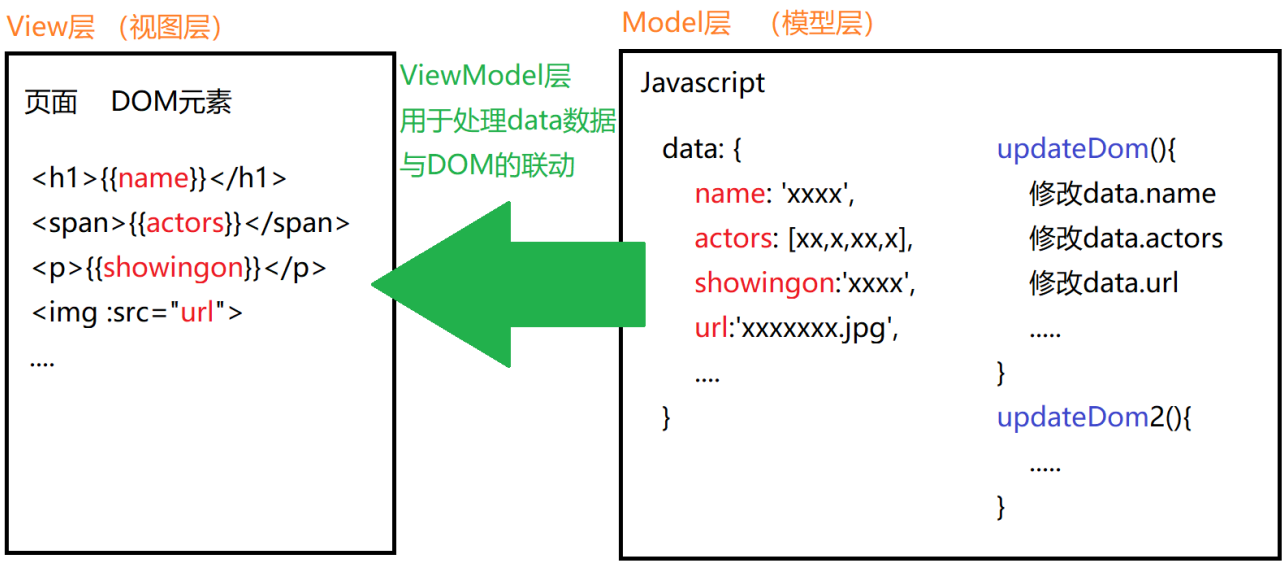
```
<div>欢迎：{{name}}</div>
<p>欢迎：{{name}}</p>
<span>欢迎：{{name}}</span>
<i>欢迎：{{name}}</i>
<u>欢迎：{{name}}</u>
......
```

```
new Vue({
    data: {
        name: '亮亮'
    }
})
```

vue的设计思想，**不再是**先找DOM元素，然后调用方法，修改DOM。而是使用 {{}} 语法将页面内容与一个js变量绑定，如果需要修改DOM时，只需要修改绑定的变量即可。

MVVM 架构

vue 框架的底层设计遵循 MVVM 架构。



开发 vue 项目的模式

1. 在 html 中引入 vue.js，通过 new Vue() 来管理 DOM。
2. 使用 Vue 脚手架来开发 vue 项目。 通过 vue 脚手架环境可以方便的创建一个通用的 vue 项目框架模板，在此基础之上开发 vue 项目更加便捷。适合工程化开发。

常见的 Vue 脚手架环境： VueCLI 。需要先安装及配置环境，才可以通过脚手架创建生成项目包。

安装配置 VueCLI，通过VueCLI创建项目包，详情参见：

```
VueCLI_intallation_guide.pdf
```

一旦项目创建成功，即可通过命令，启动脚手架项目，提供静态页面的访问。

VueCLI 项目包很大，大多数文件都在 node_modules 中，所以以后传代码时不会传递 node_modules 文件夹。大家下载项目后由于缺少模块依赖，所以无法运行项目，需要在项目根目录中，执行命令，安装所以依赖后，才可以使用：

```
# 根据当前目录下package.json中的依赖，下载相关依赖包生成node_modules
npm install
# npm install的简写
npm i
```

进入项目的目录中，执行命令，启动脚手架项目：

```
npm run serve
```

访问脚手架首页：

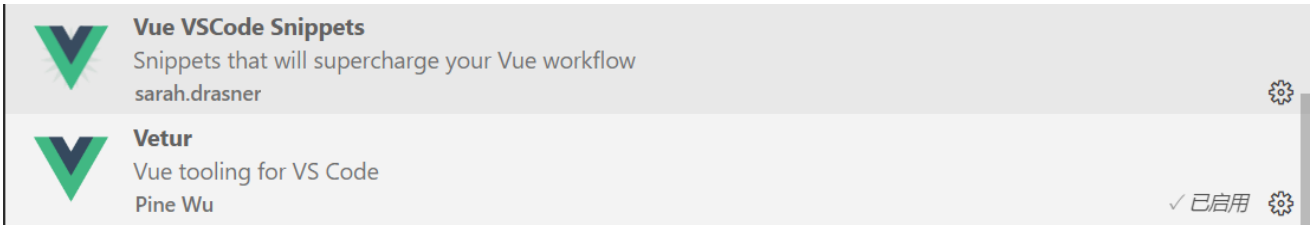
```
http://localhost:8080/
```

脚手架的运行过程： 当脚手架启动时，访问 `http://localhost:8080`，将会打开 `public/index.html`，并且在该网页中加载运行 `src/main.js` 里的代码。将会创建 `Vue` 对象，通过 `Vue` 对象来管理 `index.html` 中的 `#app` 内容的显示。初始化状态下，默认将 `App.vue` 组件中的内容渲染到 `#app` 中，从而看到页面效果。

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

安装一些 `vscode` 开发 `vue2` 项目时的插件：



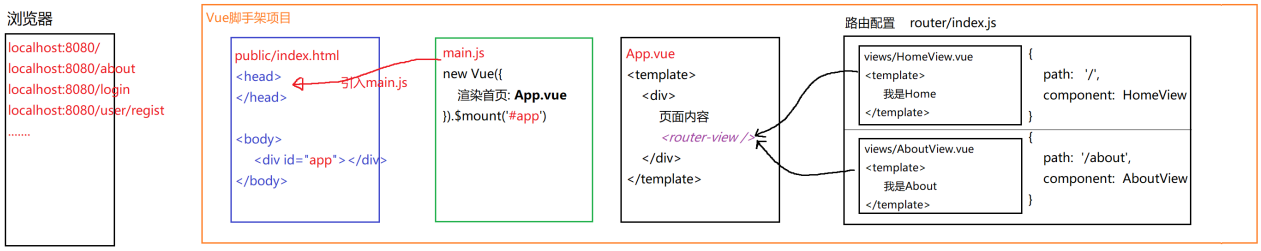
`VueCLI` 脚手架中路由系统（`VueRouter`）的设计与使用

平时在开发项目的过程中，通常需要访问不同的请求路径，从而看到不同的页面内容。`VueCLI` 脚手架中使用 `VueRouter` 来管理路由系统，最终实现效果：访问不同的路由地址，可以看到相应的页面内容。例如：

```
当访问：http://localhost:8080/          看到HomeView.vue
当访问：http://localhost:8080/about     看到About.vue

// 案例：
当访问：http://localhost:8080/home      看到HomeView.vue

// 案例：
设计路由：
当访问：http://localhost:8080/          看到HomeView.vue
当访问：http://localhost:8080/home      也能看到HomeView.vue
```



案例：写一个普通vue组件：`views/Index.vue`。

当访问：`http://localhost:8080/index`，看到这个组件页面。

课堂练习：

```
访问：http://localhost:8080/login    看到views/Login.vue 登录页面。
访问：http://localhost:8080/register    看到views/Register.vue 注册页面。
访问：http://localhost:8080/detail    看到views/Detail.vue 美食详情页面。
.......
```

`vue`文件的语法

在脚手架项目中，每一个`vue`文件都称为一个**组件（`Component`）**。一个组件封装了页面中的局部内容（包括页面结构、元素的样式、事件功能）。这样就需要研究学习一下`.vue`文件的写法，来搞定项目开发过程中的语法细节。

```
<!-- 快捷键  vbase -->
<template>
  <div>
    <h1>我是首页：Index.vue</h1>
  </div>
</template>

<script>
export default {};
</script>

<style lang="scss" scoped></style>
```

`template`用于定义当前组件的页面结构。定义的这些页面结果最终会被挂载到`#app`上。

注意：`template`中的内容有且仅有一个根元素。

`script`部分用于定义当前组件的`js`脚本代码。在`script`中可以通过`data(){}`来提供当前组件所需要的数据。也可以通过：`methods: {}`来提供当前组件所需要的方法。

`style`部分用于定义当前组件中标签元素的`css`样式。

```
<style lang="scss" scoped>  </style>
```

1. `lang="scss"`：当前`style`段支持`scss`编译。
2. `scoped`：一旦`style`标签上有`scoped`属性，则在此`style`段中定义的`css`样式，只针对当前组件生效（原理是当前组件中每一个元素都会添加一个组件唯一的随机属性：`data-v-a85a8aaa`，使用属性选择器代替普通选择器就可以仅仅选出当前组件中的元素，而不会影响其他组件中的元素的样式）。如果没有`scoped`属性，则定义的样式每一个组件都会应用。

案例：新建组件：`views/Basic.vue`访问：`/basic`时，看到该组件。

经典错误集合

```
[Vue warn]: Property or method "clickMe" is not defined on the instance but referenced during render. Make sure that this property is reactive, either in the data option, or for class-based components, by initializing the property. See: https://v2.vuejs.org/v2/guide/reactivity.html#Declaring-Reactive-Properties.

(found in <Root>)
```
vue.js:5108

```
[Vue warn]: Invalid handler for event "click": got undefined

(found in <Root>)
```
vue.js:5108

属性或方法`clickMe`并没有在vue对象中定义。要么在data段中声明，要么定义成组件的成员。

`Module not found: Error: Can't resolve './views/Basic.vue' in 'D:\code2209\01_Vue\day02\demo\vue-project\src\router`

模块找不到错误。好好检查代码中编写的这个路径是否可以找到对应的组件。当前这个错误就是找不到这个组件（`'./views/Basic.vue'`）而出的错误。

`vue`的常用指令

当`vue`加载`template`时，若发现元素的属性是`v-`前缀，就会当做是`vue`指令来进行处理。常见的指令有：

1. `v-on` 绑定事件。
2. `v-bind` 绑定属性。
3. `v-show` 动态显示元素。
4. `v-text`
5. `v-html`
6. `v-pre`
7. `v-for`
8. `v-if` `v-else-if` `v-else`
9. 等...

案例：访问地址：`/direct` 看到 `views/Direct.vue`，测试vue的常用指令。

`v-if` `v-else-if` `v-else`

根据指令值的取值（`true`、`false`）来判断是否输出当前元素：

```
<span v-if="age<18">未成年(18岁以下)</span>
<span v-else-if="age<=35">青年(18岁-35岁)</span>
<span v-else-if="age<=60">中年(36岁-60岁)</span>
<span v-else>老年(61岁以上)</span>
```

上述4个 `span`，仅输出一个。

面试题：

请说出 `v-if` 与 `v-show` 的区别。

`v-if` 指令用于定义元素是否编译输出。若为false，则页面中将不会出现该dom元素。`v-show` 则使用 `css` 的方式是元素隐藏或显示。仅仅只是通过 `css` 来控制页面的显示状态。

如果遇到了一个业务需求，需要频繁的切换元素的显示与隐藏，则适合使用v-show而不是v-if，因为v-if会动态新增、删除dom元素，在频繁切换的业务场景下性能不如v-show（css级别的显示与隐藏，并不会过多的更新dom树）。

`v-for` 指令

`v-for` 指令用于循环输出元素。例如：

```
<p v-for="item in 10">一段话...</p>
```

上面的写法，类似于：

```
for(let i=0; i<10; i++){
   let item = i+1
   输出: <p>一段话...{{item}}</p>
}
```

案例：新建一个组件：`VFor.vue`，当访问：`/vfor` 时看到这个组件。

还可以遍历数组：

```
data() {
    return {
        hobby: ['玩单杠', '摊煎饼', '背麻袋', '健身', '游泳', '瑜伽']
    }
},
```

```
<span class="tag" v-for="(item, i) in hobby" :key="item">
    {{ item }}
</span>
```

上述写法，类似于：

```
for(let i=0; i<hobby.length; i++){
   let item = hobby[i]
   输出：`<span class="tag">${item}</span>`
}
```