# 《Vue 前端开发》

> 讲师：杜Sir

## 一、认识Vue.js

### 1. Vue介绍

- Vue.js(简称Vue) 是一套用于构建用户界面的渐进式前端框架。
- Vue.js 核心实现：
  - 响应式的数据绑定:当数据发生改变，视图可以自动更新，不用关心DOM操作，而专心数据操作。
  - 可组合的视图组件:把视图按照功能切分成若干基本单元，可维护，可重用，可测试等特点。
  - 响应式，双向数据绑定，即MVVM。是指数据层（Model）-视图层（View）-数据视图（ViewModel）的响应式框架。
- vue2语法

```js
// src/components/UserRepositories.vue

export default {
  components: { RepositoriesFilters, RepositoriesSortBy, RepositoriesList },
  props: {
    user: {
      type: String,
      required: true
    }
  },
  data () {
    return {
      repositories: [], // 1
      filters: { ... }, // 3
      searchQuery: '' // 2
    }
  },
  computed: {
    filteredRepositories () { ... }, // 3
    repositoriesMatchingSearchQuery () { ... }, // 2
  },
  watch: {
    user: 'getUserRepositories' // 1
  },
  methods: {
    getUserRepositories () {
      // 使用 `this.user` 获取用户仓库
    }, // 1
    updateFilters () { ... }, // 3
  },
  mounted () {
    this.getUserRepositories() // 1
  }
}
```

- vue3语法

```js
// src/components/UserRepositories.vue `setup` function
import { fetchUserRepositories } from '@/api/repositories'
import { ref, onMounted, watch, toRefs, computed } from 'vue'

// 在我们的组件中
setup (props) {
  // 使用 `toRefs` 创建对 props 中的 `user` property 的响应式引用
  const { user } = toRefs(props)

  const repositories = ref([])
  const getUserRepositories = async () => {
    // 更新 `props.user` 到 `user.value` 访问引用值
    repositories.value = await fetchUserRepositories(user.value)
  }

  onMounted(getUserRepositories)

  // 在 user prop 的响应式引用上设置一个侦听器
  watch(user, getUserRepositories)

  const searchQuery = ref('')
  const repositoriesMatchingSearchQuery = computed(() => {
    return repositories.value.filter(
      repository => repository.name.includes(searchQuery.value)
    )
  })

  return {
    repositories,
    getUserRepositories,
    searchQuery,
    repositoriesMatchingSearchQuery
  }
}
```

官网：https://v3.cn.vuejs.org/

## 2. 引入Vue.js

- 参考文档:https://v3.cn.vuejs.org/guide/installation.html
- 在HTML中以CDN包的形式导入

  ```
  <script src="https://unpkg.com/vue@3"></script>
  ```

- 下载JS文件保存到本地再导入
- 使用npm安装

  ```
  npm install @vue/cli@4.5.12
  ```

- 使用官方VueCli脚手架构建项目（不建议新手直接使用）

  ```
  vue create vue-demo
  ```

- Hello World示例：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <!-- 引入变量 -->
        {{ message }}
    </div>

    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                //变量名和值
                const message = ref('Hello Vue!!')
                //返回出去
                return {
                    message
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 3. 声明式渲染

- Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="counter">
        Counter: {{ counter }}
    </div>
```

```
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const counter = ref(5)
                return {
                    counter
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#counter') //绑定元素
    </script>
</body>
</html>
```

- 现在数据和DOM已经被建立了关联，所有东西都是**响应式**的，可通过下面示例确认：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="counter">
        Counter: {{ counter }}
    </div>

    <script type="text/javascript">
        const { ref, onMounted } = Vue
        const HelloVueApp = {
            setup() {
                const counter = ref(5)
                //生命周期钩子
                onMounted(() => {
                    //定时器
                    setInterval(() => {
                        counter.value ++
                    }, 1000)
                })

                return {
                    counter
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#counter') //绑定元素
```

```
    </script>
</body>
</html>
```

## 4. 模板语法

- Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层组件实例 的数据。所有 Vue.js 的模板都是合法的 HTML，所以能被遵循规范的浏览器和 HTML 解 析器解析。
- 数据绑定最常见的形式就是使用“双大括号” 语法在HTML中插入文本：

  `<span>Message: {{ msg }}</span>`
- {{msg}}将被替代对应组件实例中msg属性的值。无论何时，绑定的组件实例上msg属性发 生改变，插值处内容都会更新。

## 5. Vue3响应式

- 解决了：Vue2中新增属性，删除属性(对象)，界面不会更新的问题。
- 解决了：Vue2中直接通过下标修改数组，界面不会刷新的问题。

### 5.1 ref函数

- 作用：定义一个响应式的数据。
- 语法：`let a = ref('Hello Vue!')`
- 创建一个包含响应式数据的引用对象（reference对象）。
- JS中操作数据：`a.value`
- 模板中读取数据，不需要`.value`，直接：`<div>{{ a }}</div>`
- 接收的数据可以是基本类型，也可以是对象类型。
- 基本类型的数据：响应式依然是靠Object.defineProperty()的get与set完成的。
- 对象类型的数据：内部使用了Vue3中的一个新函数：reactive函数

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <!-- 引入变量 -->
        {{ message }}
    </div>

    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                //变量名和值
                const message = ref('Hello Vue!!')
```

```
                console.log(message)
                console.log(mesage.value)
                //返回出去
                return {
                    message
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 5.2 reactive函数

- 作用：定义一个对象类型的响应式数据（基本类型数据不能用，要用ref函数）。
- 语法：const 代理对象=reactive(原函数)，接收一个对象或数组，返回一个代理对象（proxy对象）。
- reactive定义的响应式数据是深层次的。
- 内部基于ES6的Proxy实现，通过代理对象操作源内部数据进行操作。
- 用法与ref创建的代理对象类似，不过操作时不需要再加 .value 了

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <!-- 引入变量 -->
        {{ student }}
        <br>
        {{ student.name }}
    </div>

    <script type="text/javascript">
        const { reactive } = Vue
        const HelloVueApp = {
            setup() {
                //变量名和值
                const student = reactive({
                    name: "zhangsan",
                    age: 12,
                    score: 70
                })
                console.log(student)
                //返回出去
                return {
```

```
                    student
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

# 二、Vue常用指令

## 1. 指令介绍

- **指令**:带有 v- 前缀的特殊属性。
- **指令的作用**:当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

## 2. v-text

- v-text作用与双大花括号作用一样，将数据填充到标签中。但没有闪烁问题！

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p v-text="msg"></p>
        <p>{{ msg }}</p>
    </div>

    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref('Hello Vue!!')
                return {
                    msg
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
```

```
</html>
```

## 3. v-html

- 某些情况下，从服务端请求的数据本身就是一个HTML代码，如果用双大括号会将数据解 释为普通文本，而非HTML代码，为了输出真正的HTML，需要使用 v-html 指令：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        {{ msg }}
        <br>
        <span v-html="msg"></span>
    </div>

    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref("<span style='color: red'>Hello Vue!!</span>")
                return {
                    msg
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 4. v-on

- 在前端开发中，我们经常监听用户发生的事件，例如点击、拖拽、键盘事件等。 在Vue中如何监听事件呢?使用 v-on 指令
- v-on： 冒号后面是event参数，例如click、change

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
```

```
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p>点击次数: {{ counter }}</p>
        <button type="button" v-on:click="counter++">按钮</button>
        <br>
        <a type="text" id="fname" v-on:mouseout="counter++">鼠标离开</a>
    </div>

    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const counter = ref(0)
                return {
                    counter
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 5. v-bind

- 用于动态绑定一个或多个属性值，或者向另一个组件传递props值(这个后面再介绍)
- 应用场景:图片地址src、超链接href、动态绑定一些类、样式等等

### 5.1 绑定超链接

- v-bind 指令后接收一个参数，以冒号分割。
- v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <a v-bind:href="url">百度</a>
    </div>

    <script type="text/javascript">
```

```
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const url = ref("http://www.baidu.com")
                return {
                    url
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 5.2 绑定Class

- 操作元素(标签)的 class 和 style 属性是数据绑 定的一个常见需求。
- 例如希望动态切换class，为div显示不同背景颜色

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
    <style>
        .test {
            width: 200px;
            height: 200px;
            background: grey;
        }
        .active {
            background: orange;
        }
    </style>
</head>
<body>
    <div id="hello-vue">
        <!-- active这个calss存在与否取决于数据属性isActive -->
        <div v-bind:class="{active: isActive}" class="test"></div>
        <button type="button" v-on:click="btn()">增加样式</button>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const isActive = ref(false)
                //定义方法
```

```
                function btn() {
                    if (isActive.value) {
                        isActive.value = false
                    } else {
                        isActive.value = true
                    }
                }

                return {
                    isActive,
                    btn
                }
            }
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 5.3 绑定Style

- v-bind:style 的对象语法看着非常像 CSS，但其实是一个 JavaScript 对象。
- 可以使用v-bind在style样式中传递样式变量。
- 使用时需要将css样式名中带”-“的转成驼峰命名法，如font-size，转为fontSize

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <div v-bind:style="{background: background, fontSize: fontSize + 'px'}">
            Hello vue!!
        </div>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const background = ref('orange')
                const fontSize = ref('24')

                return {
                    background,
                    fontSize,
```

```
                }
            },
        }
        Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 5.4 指令缩写

- v- 前缀作为一种视觉提示，用来识别模板中 Vue 特定的 属性。 但对于一些频繁用到的指令来说，就会感到使用繁琐。
- 因此，Vue 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写：
- v-bind缩写

```
<!-- 完整语法 -->
<a v-bind:href="url"> ... </a>

<!-- 缩写 -->
<a :href="url"> ... </a>

<!-- 动态参数的缩写 -->
<a :[key]="url"> ... </a>
```

- v-on缩写

```
<!-- 完整语法 -->
<a v-on:click="doSomething"> ... </a>

<!-- 缩写 -->
<a @click="doSomething"> ... </a>

<!-- 动态参数的缩写 -->
<a @[event]="doSomething"> ... </a>
```

# 三、Vue常用属性

## 1. setup函数

- setup是vue3中的一个**全新的配置项**，值为一个**函数**。
- setup是所有**CompositionAPI**（组合API）的基础，组件中所用到的**数据、方法**等都需要在setup中进行配置。
- setup中的两种返回值：
  - 若返回一个对象，则对象中的属性，方法，在模板中都可以使用。
  - 若返回一个渲染函数：可以定义渲染内容。

```
import { h } from 'vue'
...
setup() {
    ...
    return () => h('h1','学习')
}
```

- setup执行的时机：在 beforeCreate 之前执行一次， this 是 undefine 。
- setup的参数
  - props：值为对象，包含：组件外部传递过来，且组件内部声明接收了的属性。
  - context：上下文对象
    - attrs：值为对象，包含组件外部传递过来，但没有在props配置中声明的属性。
    - slots：收到的插槽内容，想到相当于 this.$slots 。
    - emit：分发自定义事件的函数，相当于 this.$emit 。
- 注意点：
  - 尽量不要与Vue2配置混用。
    - Vue2配置（data,methos, computed...）中可以访问到 setup 中的属性，方法。
    - 但setup中不能访问到vue2配置（data,methos,computed...）。
    - 如果重名，setup中的优先。
  - setup不能是 async 函数，因为返回值不再是 return 的对象，而是 promise ，模板中看不到 return 对象中的属性。

## 2．方法

- 使用function关键字定义的数据处理函数。
- 定以后在return中返回，才能在模板中使用。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p>{{ count }}</p>
        <button type="button" @click="increment()">增加</button>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const count = ref(1)

                function increment() {
                    count.value ++
```

```
                }

                return {
                    count,
                    increment
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
        console.log(vm.count)  // 1
        vm.increment()  //调用方法
        console.log(vm.count)  // 2
    </script>
</body>
</html>
```

## 3. 计算属性

- **计算属性(computed):**根据所依赖的数据动态显示新的计算结果。
- 示例:需要在{{ }}里添加计算再展示数据，例如统计分数
- 数值计算一般用法:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <!-- 数值计算一般用法 -->
    <div id="hello-vue">
        <!-- 支持JS表达式 -->
        <span>总分: {{ math + chinese + english }}</span>
    </div>
    <script type="text/javascript">
        const { ref,computed } = Vue
        const HelloVueApp = {
            setup() {
                const math = ref(90)
                const chinese = ref(88)
                const english = ref(62)

                return {
                    math,
                    chinese,
                    english
                }
```

```
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

- computed计算属性用法:
- 小结:计算属性一般就是用来通过其他的数据算出一个新数据，而且它有一个好处就是， 它把新的数据缓存下来了，当其他的依赖数据没有发生改变，它调用的是缓存的数据，这 就极大的提高了我们程序的性能。而如果写在function里，数据根本没有缓存的概念，所 以每次都会重新计算。这也是为什么不用function的原因!

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <!-- 数值计算一般用法 -->
    <div id="hello-vue">
        <!-- 支持JS表达式 -->
        <span>总分: {{ sum }}</span>
    </div>
    <script type="text/javascript">
        const { ref,computed } = Vue
        const HelloVueApp = {
            setup() {
                const math = ref(90)
                const chinese = ref(88)
                const english = ref(62)

                const sum = computed(() => {
                    return math.value + chinese.value + english.value
                })

                return {
                    sum
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 4. 监听属性

- **监听属性(watch):**是一个观察动作，监听data数 据变化后触发对应函数，函数有newValue(变化之后 结果)和oldValue(变化之前结果)两个参数。 当需要在数据变化时执行异步或开销较大的操作时， 这个方式是最有用的。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p>消息: {{ msg }}</p>
        <p>观察的消息: {{ watchMsg }}</p>
        <button type="button" @click="btn()">按钮</button>
    </div>
    <script type="text/javascript">
        const { ref, watch } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref('hello vue!')
                const watchMsg = ref()
                //方法
                 //点击按钮执行该函数，msg重新赋新值，模拟msg值改变
                function btn() {
                    msg.value = 'hello go!'
                }
                //监听
                watch(msg, (newVal, oldVal) => {
                    console.log(newVal, oldVal)
                    watchMsg.value = newVal
                })

                return {
                    msg,
                    watchMsg,
                    btn
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

# 四、Vue流程控制

## 1. v-if、v-else-if、v-else

- 示例：判断一个元素是否显示
- v-if 指令将根据表达式 seen 的值的真假来插入或移除 <p> 元素。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet" /> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p v-if="seen">现在你看到我了</p>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const seen = ref(true)
                return {
                    seen
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

- 示例：添加一个v-else

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet" /> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p v-if="seen">现在你看到我了</p>
        <p v-else>看不到我了</p>
    </div>
```

```
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const seen = ref(false)
                return {
                    seen
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

- v-if指令必须将它添加到一个元素上。如果想切换多个元素呢?

  此时可以把一个template元素当做不可见的包裹元素，并在上面使用v-if 。最终的渲染 结果将不包含template元素。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <template v-if="seen">
            <h1>标题</h1>
            <p>这是第一个段落。</p>
            <p>这是第二个段落</p>
        </template>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const seen = ref(true)
                return {
                    seen
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

- v-else-if多分支

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <div v-if="type === 'A'">
            <p>A</p>
        </div>
        <div v-else-if="type === 'B'">
            <p>B</p>
        </div>
        <div v-else>
            <p>不是A和B! </p>
        </div>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const type = ref('C')
                return {
                    type
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 2. v-show

- v-show:另一个用于条件性展示元素的指令。
- 与v-if不同的是，v-show的元素始终会被渲 染并保留再DOM中，所以v-show只是简单地切换元素的display CSS属性，display: none 。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
```

```
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p v-show="seen">现在你看到我了</p>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const seen = ref(true)
                return {
                    seen
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 3. v-for

- 可以用 `v-for` 指令基于一个数组来渲染一个列表。
- `v-for` 指令需要使用 `item in items` 形式 的特殊语法，其中 `items` 是源数据数组，而 `item` 则是被迭代的数组元素的别名。
- 示例：遍历数组

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <li v-for="(c, i) in myArray">
            {{ i }} - {{ c }}
        </li>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const myArray = ref([
                    '主机',
                    '显示器',
```

```
                    '键盘'
                ])
                return {
                    myArray
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

- 示例：遍历对象

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <li v-for="(v, k) in myObject">
            {{ k }} - {{ v }}
        </li>
    </div>
    <script type="text/javascript">
        const { reactive } = Vue
        const HelloVueApp = {
            setup() {
                const myObject = reactive({
                    host: '主机',
                    displayer: '显示器',
                    keyboard: '键盘'
                })
                return {
                    myObject
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 4. v-for: 就地更新

- 当 Vue 正在更新使用 v-for 渲染的元素列表时，它默认使用“就地更新”的策略。如果数 据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个 元素，并且确保它们在每个索引位置正确渲染。
- 为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你 需要为每项提供一个唯一的 key 属性：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <ul>
            <li v-for="(item,index) in myArray" :key="item.id">
                {{item.id}}
                <input type="text" :placeholder="item.name">
                <button v-on:click="btn(index)">删除</button>
            </li>
        </ul>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const myArray = ref([
                    {id:'1', name: '主机'},
                    {id:'2', name: '显示器'},
                    {id:'3', name: '键盘'}
                ])

                function btn(i) {
                    myArray.value.splice(i,1)
                    console.log(myArray.value)
                }

                return {
                    myArray,
                    btn
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

## 5. v-for: 选择列表案列

- 获取用户选择并赋值另一个变量再实时展示:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <select @change="selectComputer($event)">
            <option value="None">未选择</option>
            <option v-for="row in computer" :key="row.id" :value="row.id"
:label="row.name"></option>
        </select>
        <p>当前选择主机ID: {{ selectComputerId }}</p>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const computer = ref([
                    {id:'1', name: '主机1'},
                    {id:'2', name: '主机2'},
                    {id:'3', name: '主机3'}
                ])
                const selectComputerId = ref()
                //方法
                function selectComputer(event) {
                    console.log(event) //获取该事件的事件对象
                    selectComputerId.value = event.target.value //获取事件的值
                    if (selectComputerId.value == 'None') {
                        selectComputerId.value = '未选择! '
                    }
                }
                return {
                    computer,
                    selectComputerId,
                    selectComputer
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
```

```
</body>
</html>
```

# 五、Vue数据双向绑定

## 1. v-model

- 双向数据绑定:通过前面学习知道Vue是数据驱动的，数据驱动有一个精髓之处是数据双向绑定， 即当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化。
- `v-model`指令其实是一个语法糖，背后本质上包含`v-bind`和`v-on`两个操作。

```
<input v-model="sth" />
//  等同于
<input :value="sth" @input="sth = $event.target.value" />
```

- 详解
  - `$event` 指代当前触发的事件对象。

    `$event.target` 指代当前触发的事件对象的dom

    `$event.target.value` 就是当前dom的value值

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <input type="text" :value="msg" @input="input($event)">
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref('hello vue!')
                function input(event) {
                    console.log(msg.value)
                    msg.value = event.target.value
                    console.log(event.target.value, msg.value)
                }
                return {
                    msg,
                    input
                }
```

```
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 1.1 v-model：表单输入

- v-model指令提供表单输入绑定，可以在 <input>、<textarea> 及 <select> 元素上创建双向 数据绑定。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet" /> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <input type="text" v-model="msg"></input>
        <p>{{ msg }}</p>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref('hello vue!')
                return {
                    msg
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 1.2 v-model：单选框

- **单选框(radio)**：单个选择结果绑定到一个v-model的值中

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet" /> -->
```

```
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <input type="radio" value="Go" v-model="msg">Go</input>
        <input type="radio" value="Vue" v-model="msg">Vue</input>
    </div>
    <script type="text/javascript">
        const { ref, watch } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref()

                watch(msg, (newVal, oldVal) => {
                    console.log(msg.value, newVal, oldVal)
                })

                return {
                    msg
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 1.3 v-model：多选框

- **多选框(checkbox)**：多个选择结果绑定到一个 v-model 的值中

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet" /> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <input type="checkbox" v-model="selected" value="游泳">游泳</input>
        <input type="checkbox" v-model="selected" value="健身">健身</input>
        <input type="checkbox" v-model="selected" value="旅游">旅游</input>
    </div>
    <script type="text/javascript">
        const { ref } = Vue
        const HelloVueApp = {
            setup() {
                const selected = ref([])
```

```
                watch(selected, (newVal, oldVal) => {
                    console.log(selected.value, newVal, oldVal)
                })

                return {
                    selected
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

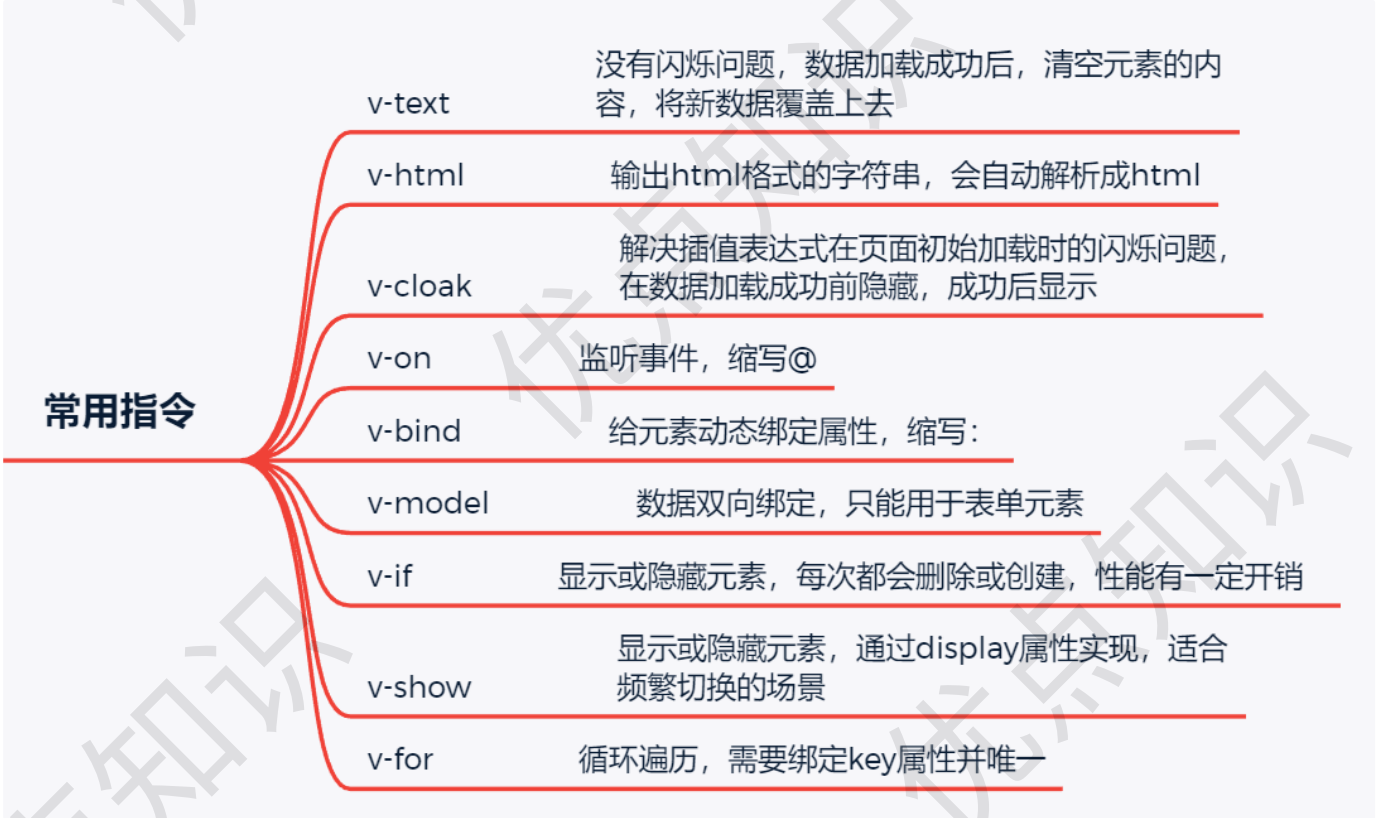
### 1.4 v-model: 登录案例

- 登录案例:获取用户输入用户名和密码

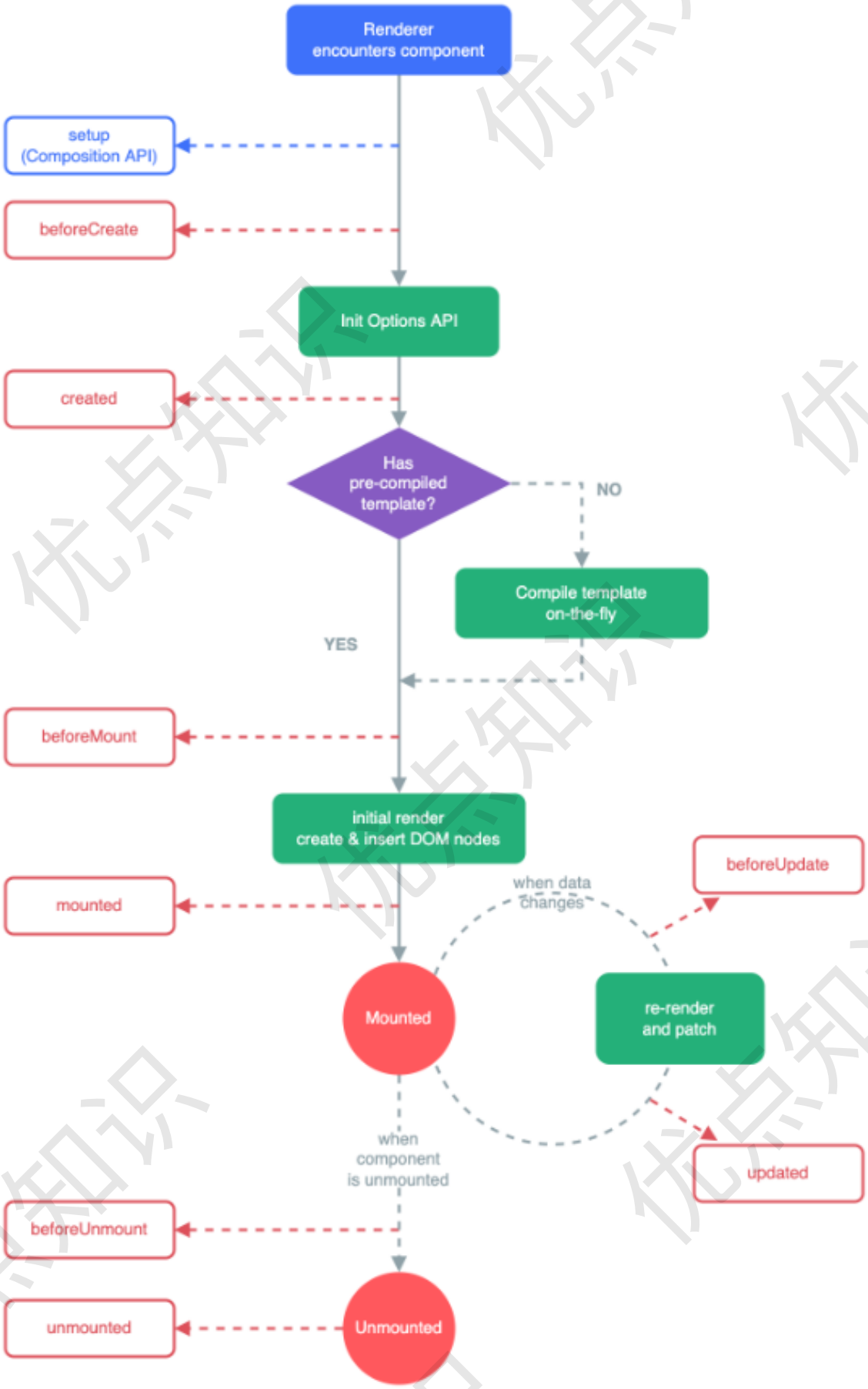
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <form action="#">
            <h1>欢迎访问管理后台</h1>
            用户名: <input type="text" v-model="form.username"></input>
            密码: <input type="password" v-model="form.password"></input>
            <button @click="loginBtn">登录</button>
        </form>
        <p style="color: red;" v-if="notice">用户名或者密码不能为空!</p>
    </div>
    <script type="text/javascript">
        const { ref, reactive } = Vue
        const HelloVueApp = {
            setup() {
                const form = reactive({
                    username: '',
                    password: ''
                })
                const notice = ref(false)

                function loginBtn() {
                    if (form.username == '' || form.password == '') {
                        notice.value = true
```

```
                    } else {
                        notice.value = false
                        console.log(form)  //获取用户名和密码，提交到服务器。。。
                    }
                }

                return {
                    form,
                    notice,
                    loginBtn
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

### 2. 常用指令总结




## 六、Vue实例生命周期钩子




- 生命周期是指Vue实例从创建到销毁的过程。
- 就是vue实例从开始创建、 初始化数据、编译模板、挂载Dom、渲染→更新→渲染、卸载等一系 列过程。

- 在vue生命周期中提供了一系列的生命周期函数，除了beforecate和created（它们被setup方法本身所取代），我们可以在setup方法中访问的API生命周期钩子有9个选项：
  - onBeforeMount - 在挂载开始之前被调用：相关的 render 函数首次被调用。
  - onMounted - 组件挂载时调用
  - onBeforeUpdate - 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。
  - onUpdated - 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。
  - onBeforeUnmount - 在卸载组件实例之前调用。在这个阶段，实例仍然是完全正常的。
  - onUnmounted - 卸载组件实例后调用。调用此钩子时，组件实例的所有指令都被解除绑定，所有事件侦听器都被移除，所有子组件实例被卸载。
  - onActivated - 被 keep-alive 缓存的组件激活时调用。
  - onDeactivated - 被 keep-alive 缓存的组件停用时调用。
  - onErrorCaptured - 当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。
- 代码示例

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>文档的标题</title>
    <!-- <link href="main.css" type="text/css" rel="stylesheet"/> -->
    <script src="https://unpkg.com/vue@3"></script>
</head>
<body>
    <div id="hello-vue">
        <p>{{ msg }}</p>
        <button @click="btn()">修改</button>
    </div>
    <script type="text/javascript">
        const { ref, onBeforeMount, onMounted, onBeforeUpdate, onUpdated } = Vue
        const HelloVueApp = {
            setup() {
                const msg = ref('hello aaa!')

                function btn() {
                    msg.value = 'hello bbb!'
                }

                onBeforeMount(() => {
                    console.log('onBeforMount')
                })
                onMounted(() => {
                    console.log('onMounted')
                })
                onBeforeUpdate(() => {
                    console.log('onBeforeUpdate')
                })
                onUpdated(() => {
                    console.log('onUpdated')
                })
```

```
                return {
                    msg,
                    btn
                }
            }
        }
        const vm = Vue.createApp(HelloVueApp).mount('#hello-vue')
    </script>
</body>
</html>
```

# 七、VueCli脚手架

- 到目前为止，已经会了Vue基本使用，但这种在HTML引用Vue.js的方式，简单的页面还是没问题 的，如果用Vue开发整个前端项目，组建Vue项目结构及配置还是比较复杂的，例如引入各种js文 件、打包上线等。
- 因此，为了提高开发效率，官方开发了VueCli脚手架快捷搭建开发环境。

## 1. Vue Cli脚手架介绍

Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供：

- 通过 @vue/cli 实现的交互式的项目脚手架。
- 通过 @vue/cli + @vue/cli-service-global 实现的零配置原型开发。
- 一个运行时依赖 (@vue/cli-service)，该依赖：
  - 可升级；
  - 基于 webpack 构建，并带有合理的默认配置；
  - 可以通过项目内的配置文件进行配置；
  - 可以通过插件进行扩展。
- 一个丰富的官方插件集合，集成了前端生态中最好的工具。
- 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

Vue CLI 致力于将 Vue 生态中的工具基础标准化。它确保了各种构建工具能够基于智能的默认配 置即可平稳衔接，这样你可以专注在撰写应用上，而不必花好几天去纠结配置的问题。

## 2. 认识NPM

在使用Vue Cli之前，需先了解一些关于NPM的知识点：

- **NPM(Node Package Manager，Node包管理器)，**存放JavaScript代码共享中心，是目 前最大的JavaScript仓库。类似于Linux yum仓库。
- 可能你会联想到Node.js，Node.js是服务端的JavaScript，类似于Gin、Django，NPM是基 于Node.js开发的软件。
- 随着Node.js兴起，生态圈的JS库都纷纷向NPM官方仓库发布，所以现在，大都是使用npm install命令来安装JS库，而不必再去它们官网下载了。
- 安装Node.js，默认已经内置npm，下载对应软件包直接安装即可。http://nodejs.cn/download/
- 常用命令介绍

| 命令 | 描述 |
| --- | --- |
| npm -v | 查看版本 |
| npm install <模块名> | 在项目目录下安装模块 |
| npm install -g <模块名> | 全局安装模块 |
| npm list -g | 查看所有全局安装的模块 |
| npm list <模块名> | 查看某个模块的版本号 |
| npm install -g <模块名>@<版本号> | 全局更新模块版本 |
| npm install --save <模块名> | 在package.json文件中写入依赖（npm5版本之前需要指定，之后版本 无需再加--save选项） |
| npm config | 管理npm的配置路径 |
| npm run serve | 运行项目 |
| npm run build | 打包项目 |

- 配置npm淘宝源（默认国外源，下载依赖较慢）：

```
#设置淘宝源
npm config set registry https://registry.npm.taobao.org --global
#查看源
npm config get registry
```

# 3. Vue Cli脚手架使用

## 3.1 Vue Cli脚手架使用步骤

- 命令安装:npm install -g @vue/cli@4.5.12
- 检查版本:vue -V
- 创建项目:vue create <项目名称>
- 选择Vue3项目
- 运行项目:npm run serve
- 访问

Home | About

# Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.

### Installed CLI Plugins

babel router eslint

### Essential Links

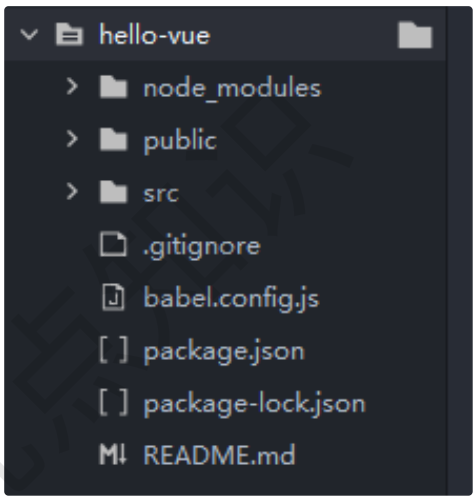
Core Docs Forum Community Chat Twitter News

### Ecosystem

vue-router vuex vue-devtools vue-loader awesome-vue



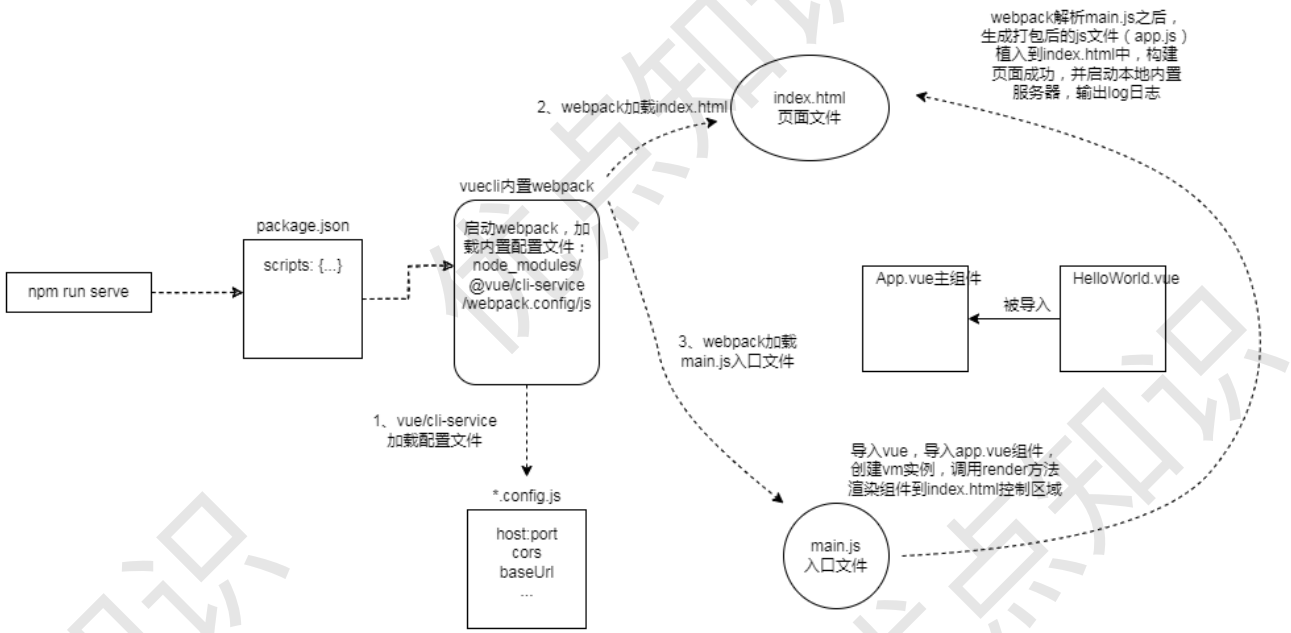
## 3.2 Vue项目目录




- 目录说明

| 目录/文件 | 说明 |
| --- | --- |
| node_modules | 项目开发依赖的一些模块，不用管 |
| public | 主要存放首页、favicon |
| src | 源码目录，这里是我们要开发的目录，基本上要做的事情都在这个目录 里。<br>里面包含了几个目录及文件：<br>- assets：放入资源，例如图片、CSS等 ⍰ components：公共组件目录<br>- routes:前端路由<br>- views:单页面组件目录<br>- App.vue：项目入口文件(根组件)<br>- main.js：项目的全局配置，在任意一个文件中都有效的 |
| .gitignore | git提交忽略文件 |
| babel.config.js | babel配置，例如es5转es6 |
| package.json | 项目配置文件。 npm包配置文件，里面定义了项目的npm脚本，依赖包等信息 |
| README.md | 项目的说明文档，markdown 格式 |

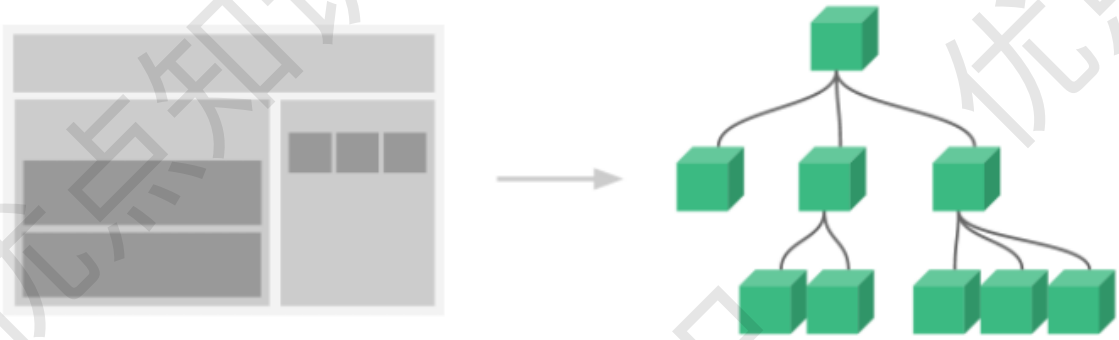
### 3.3 启动流程




# 八、Vue组件

## 1. 组件介绍

- **组件:**一段独立的，能代表页面某一个部分的代码片段，拥有自己独立的数据、JavaScript脚本、 以及CSS样式。
- 组件是可复用的Vue实例，在开发过程中可以把经常重复的功能，封装为组件，达到快捷开发的 目的。
- 组件的好处:
  - 提高开发效率
  - 方便重复使用
  - 易于管理和维护
- 通常一个应用会以一棵嵌套的组件树的形式来组织，如图所 示。
- 例如，你可能会有页头、侧边栏、内容区等组件，每个组件 又包含了其它的像导航链接、博文之类的组件。



## 2. 文件格式

- Vue单文件组件(又名*.vue文件，缩写为SFC)是一种特殊 的文件格式，它允许讲Vue组件的模板、逻辑与样式封装在单个文件中。
- 正如所见，Vue SFC 是经典的 HTML、CSS 与 JavaScript 三 个经典组合的自然延伸。每个 *.vue 文件由三种类型的顶层 代码块组成:template、script与 style
  - <template> 部分定义了组件的模板。
  - <script> 部分是一个标准的 JavaScript 模块。它应该导 出一个 Vue 组件定义作为其默认导出。
  - <style> 部分定义了与此组件关联的 CSS。

```
<template>
    <p class="greeting"> {{ greeting }} </p>
</template>

<script>
import { ref } from 'vue'
export default {
    setup() {
        const greeting = ref()
        return {
            greeting
        }
    }
}
</script>
```

```
<style scoped>
    .greeting {
        color: red;
        font-weight: bold;
    }
</style>
```

## 3. 组件使用


- 在src/components目录里开发一个组件文件(首字母大写)
- 在父组件里引用子组件 import xxx from‘xxx’
- 在默认导出里注册组件
- 在template模板里使用组件
- 写一个Test.vue组件

```
<!— src/components/Test.vue —>
<template>
    <div>
        <button>Test</button>
    </div>
</template>
```

## 4. 注册组件

- 为了能在模板中使用，这些组件必须先注册以便 Vue 能够识别。这里有两种组件的注册类 型:全局注册和局部注册。
- **局部注册**:在使用组件的vue文件中声明和使用，一般只需要解耦代码时使用
- **全局注册**:声明一次，在任何vue文件模板中使用，一般使用该组件的地方多时使用

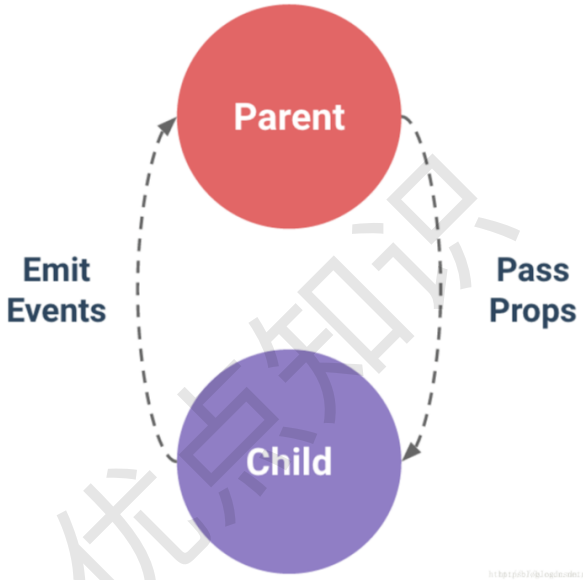
```
//main.js文件
import { createApp } from 'vue'
import App from './App.vue'
import Test from './components/Test.vue' //导入组件

const app = createApp(App)

app.component('Test', Test) //注册组件
app.mount('#app')
```

全局注册后，在任意.vue文件里可使用该组件：

```
<!-- src/App.vue -->
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <HelloWorld msg="Welcome to Your Vue.js App"/>
  <!-- 全局引入组件 -->
  <Test></Test>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

## 5. 组件传参

- 学习了组件用法，就像一种嵌套引用关系，在这个关系中，经常会涉及相互传数据的需求，即父组件传子组件，子组件传父组件。
- 父、子组件的关系可以总结为： **prop**向下传递，**emit**事件向上传递。
- 如下图所示：




- **父传子:**在默认页面中，也用到了父传子，在父组件Home.vue中给引用 的组件传入一个静态的值，子组件通过props属性接收，并在模板中使用。

```
<!— src/App.vue —>
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <!— 局部引入组件第三步：使用组件 —>
  <HelloWorld msg="Welcome to Your Vue.js App"/>
  <!— 全局引入组件 —>
  <Test></Test>
</template>

<script>
// 局部引入组件第一步：导入
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  // 局部引入组件第二步：声明组件
  components: {
    HelloWorld
  }
}
</script>

...
```

```
<!-- src/components/HelloWorld.vue -->
...
<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
...
```

- 也可以通过v-bind或者简写:动态赋值，例如:

```
<!-- src/App.vue -->
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <!-- 局部引入组件第三步：使用组件 -->
  <HelloWorld :msg="msg"/>
  <!-- 全局引入组件 -->
  <Test></Test>
</template>

<script>
// 局部引入组件第一步：导入
import HelloWorld from './components/HelloWorld.vue'
import { ref } from 'vue'
export default {
  name: 'App',
  // 局部引入组件第二步：声明组件
  components: {
    HelloWorld
  },
  setup() {
    const msg = ref("你好，这里是父组件")
    return {
      msg
    }
  }
}
</script>

...
```

- **emit**事件子传父:

```
<!-- src/components/HelloWorld.vue -->
<template>
```

```
    ...
    <button @click="btn()">按钮1</button>
  </div>
</template>

<script>
import { ref } from 'vue'
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  },
  //传递给父组件的事件
  emits: ['childMsg'],
  //props获取父传子的相关数据
  //ctx上下文，这里用到了emit方法
  setup(props, ctx) {
    const text = ref('你好！这里是子组件的数据')
    function btn() {
      //通过ctx上下文的emit方法传递数据给父组件
      ctx.emit("childMsg", text.value)
    }
    return {
      text,
      btn
    }
  }
}
</script>
```

```
<!-- src/App.vue -->
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <!-- 局部引入组件第三步：使用组件 -->
  <!-- 通过子组件的childMsg事件，绑定方法获取数据 -->
  <HelloWorld :msg="msg" @childMsg="getText"/>
  <!-- 全局引入组件 -->
  <Test></Test>
</template>

<script>
// 局部引入组件第一步：导入
import HelloWorld from './components/HelloWorld.vue'
import { ref } from 'vue'
export default {
  name: 'App',
  // 局部引入组件第二步：声明组件
  components: {
    HelloWorld
```

```
  },
  setup() {
    const msg = ref("你好，这里是父组件")
    //获取子组件数据的方法
    function getText(data) {
      console.log(data)
    }
    return {
      msg,
      getText
    }
  }
}
</script>
```

# 九、前后端数据交互：Axios

## 1. Axios 介绍

- 官方文档：http://www.axios-js.com/zh-cn/docs/
- 在前端页面展示的数据大多数都是通过访问一个API获取的，做这件事的方法有好几种，例如 jquery ajax、vue-resource、axios，而vue-resource是vue插件，但3版本不再更新，目前官方 推荐主流的axios，aixos是一个http请求库。

## 2. Axios 使用

- 安装axios：`npm install axios`
- 在组件中导入并使用

  `import axios from 'axios'`

### 2.1 GET请求

- 使用`axios.get`发起GET请求。
- 使用`.then`获取响应数据。

```
<!-- src/App.vue -->
<template>
    <div class="home">
        <img alt="Vue logo" src="./assets/logo.png" />
        <button type="button" @click="getData()">获取后端数据</button>
        {{ data }}
    </div>
</template>

<script>
// @ 是/src的别名
```

```
import axios from "axios"
import { ref } from 'vue'
export default {
    name: "Home",
    setup() {
        const data = ref()

        function getData() {
            axios.get('http://httpbin.org/get')
            .then(res => {
                data.value = res.data
            })
        }

        return {
          data,
          getData
        }
    }
}
</script>
```

- 如果打开页面就自动渲染，可以使用mounted生命周期钩子

```
<script>
// @ 是/src的别名
import axios from "axios"
import { onMounted, ref } from 'vue'
export default {
    name: "Home",
    setup() {
        const data = ref()

        function getData() {
            axios.get('http://httpbin.org/get')
            .then(res => {
                data.value = res.data
            })
        }

        onMounted(() => {
            getData()
        })

        return {
          data,
          getData
        }
    }
}
```

```
</script>
```

### 2.2 POST请求

- 登录时提交表单数据案例

```
<!-- src/App.vue -->
<template>
    <div class="home">
        <img alt="Vue logo" src="./assets/logo.png" />

        <h1>欢迎访问管理后台</h1>
        <form action="#">
            用户名: <input type="text" v-model="form.username">
            密码:  <input type="password" v-model="form.password">
            <button @click="loginBtn">登录</button>
        </form>
        <p style="color: red;" v-if="notice">用户名和密码不能为空</p>
    </div>
</template>

<script>
// @ 是/src的别名
import axios from "axios"
import { ref, reactive } from 'vue'
export default {
    name: "Home",
    setup() {
        const form = reactive({
            username: '',
            password: ''
        })
        const notice = ref(false)

        function loginBtn() {
            if (form.username == '' || form.password == '') {
                notice.value = true
                return
            } else {
                notice.value = false
            }
            axios.post('http://httpbin.org/post', form)
            .then(res => {
                console.log(res)
            })
        }

        return {
          form,
          notice,
```

```
            loginBtn
        }
    }
}
</script>
```

### 2.3 异常处理

- 很多时候我们可能并没有从 API 获取想要的数据。这可能是由于很多种因素引起的，比如 axios 调用可能由于多种原因而失败，包括但不限于：
  - API 不工作了；
  - 请求发错了；
  - API 没有按我们预期的格式返回信息。
- 可以使用catch异常处理这些问题。
- 模拟连接一个不存在的地址，前段给出提示：

```
<!-- src/App.vue -->
<template>
    <div class="home">
        <img alt="Vue logo" src="./assets/logo.png" />
        <button type="button" @click="getData()">获取后端数据</button>
        {{ data }}
        <p v-if="error" style="color: red;">连接服务器失败，请稍后再试！ </p>
    </div>
</template>

<script>
// @ 是/src的别名
import axios from "axios"
import { ref } from 'vue'
export default {
    name: "Home",
    setup() {
        const data = ref()
        const error = ref(false)

        function getData() {
            axios.get('http://httpbin.org/get11')
            .then(res => {
                data.value = res.data
            })
            .catch(res => {
                console.log(res)
                error.value = true
            })
        }

        return {
          data,
```

```
            error,
            getData
        }
    }
}
</script>
```

### 2.4 全局默认值

- 在实际开发中，几乎每个组件都会用到axios发起数据请求，如果每次都填写完整的请求路径，不利于后期维护。这时可以设置全局axios默认值。
- 添加axios全局配置

```
//main.js文件
import { createApp } from 'vue'
import App from './App.vue'
import Test from './components/Test.vue' //导入组件
import axios from 'axios'

const app = createApp(App)
axios.defaults.baseURL = 'http://httpbin.org'
axios.defaults.timeout = 5000
app.component('Test', Test) //注册组件
app.mount('#app')
```

- 将axios请求的url改为路径

```
axios.get('/get')
```

### 2.5 自定义实例默认值

- 有时候服务端接口有多个地址，就会涉及请求的域名不同、配置不同等，这时自定义实例可以很好解决。

(1) 创建src/request/index.js文件，定义实例

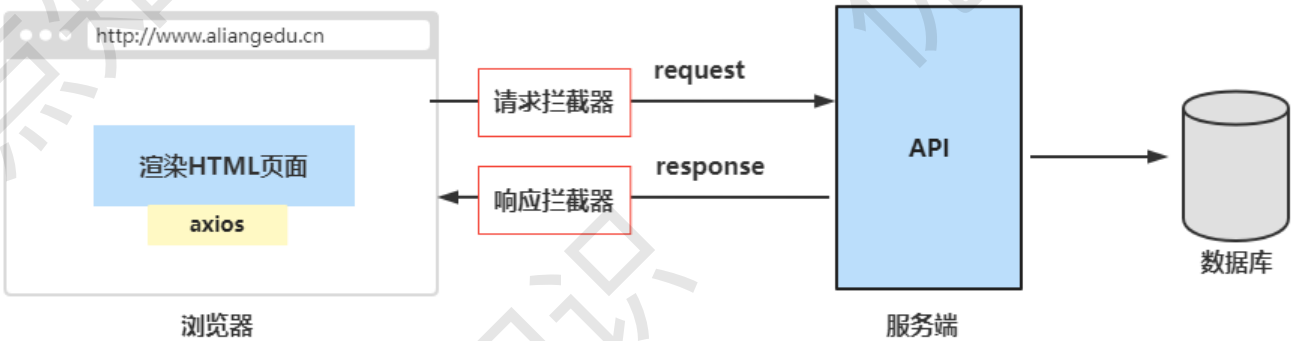
```
import axios from 'axios'

const instance = axios.create({
    baseURL: 'http://www.httpbin.org',
    timeout: 3000,
    // headers: {'X-Custom-Header': 'foobar'}
})
// 暴露函数
export default instance
```

(2) 组件使用

```
<!-- src/App.vue -->
<template>
    <div class="home">
        <img alt="Vue logo" src="./assets/logo.png" />
```

```
        <button type="button" @click="getData()">获取后端数据</button>
        {{ data }}
        <p v-if="error" style="color: red;">连接服务器失败，请稍后再试！</p>
    </div>
</template>

<script>
// @ 是/src的别名
import httpClient from "@/request"
import { ref } from 'vue'
export default {
    name: "Home",
    setup() {
        const data = ref()
        const error = ref(false)

        function getData() {
            httpClient.get('/get')
            .then(res => {
                data.value = res.data
            })
            .catch(res => {
                console.log(res)
                error.value = true
            })
        }

        return {
          data,
          error,
          getData
        }
    }
}
</script>
```

### 2.6 拦截器



- 拦截器可以拦截每一次请求和响应，然后进行相应的处理。
- 请求拦截应用场景：
  - 发起请求前添加header
- 响应拦截应用场景：
  - 统一处理API响应状态码200或非200的提示消息
  - 统一处理catch异常提示信息
- 示例代码

```
// src/request/index.js
import axios from 'axios'

const instance = axios.create({
    baseURL: 'http://www.httpbin.org',
    timeout: 3000,
    // headers: {'X-Custom-Header': 'foobar'}
})

//拦截器，请求拦截
instance.interceptors.request.use(config => {
    //在请求被发送之前做些什么
    //例如添加请求头
    config.headers['Test-Header'] = '123456'
    return config
},
error => {
    //处理请求错误
    return Promise.reject(error)
})

//拦截器，响应拦截
instance.interceptors.response.use(response => {
    if (response.data.code =/= 200) {
        alert("数据请求失败！") //这里应根据后端返回的消息显示
    }
    return response
},
error => {
    //处理响应错误(catch)
    alert("请求服务端接口错误！")
    return Promise.reject(error)
})
// 暴露函数
export default instance
```

# 十、Vue路由：Vue Router

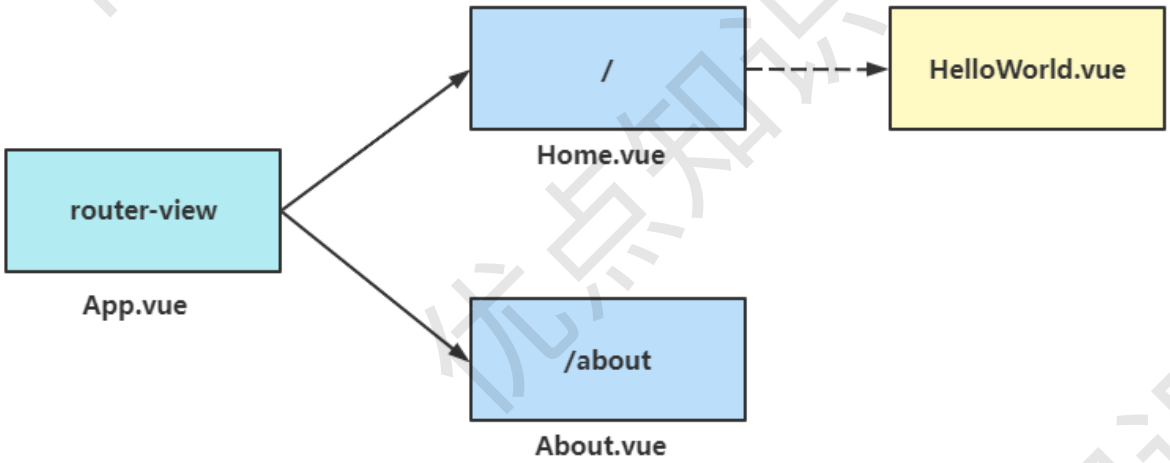
## 1. Vue Router介绍

- **Vue Router** 是 Vue.js (opens new window)官方的路由管理器。
- 核心深度集成，包含的功能有：
  - 嵌套的路由/视图表
  - 模块化的、基于组件的路由配置
  - 路由参数、查询、通配符
  - 基于 Vue.js 过渡系统的视图过渡效果
  - 细粒度的导航控制

## 2. Vue Router安装

- `npm install vue-router`
- 创建src/router/index.js文件及目录，之后前端的路由都将维护在此文件中

## 3. 使用流程




- 开发页面（组件）
- 定义路由，src/router/index.js

```
import {createRouter, createWebHashHistory} from "vue-router"
import Home from "@/views/Home.vue" // 导入组件方式1：先导入，下面引用

//1.定义路由对象
const routes = [
    {
        path: "/home", //访问路径
        name: "Home", //路由名称
        component: Home  //引用组件
    },
    {
        path: "/about",
```

```
        name: "About",
        // 导入组件方式2：当路由被访问时才会加载组件（惰性）
        component: () => import("@/views/About.vue")
    }
]

//2.创建路由实例并传递上面路由对象routes
const router = createRouter({
    history: createWebHashHistory(),
    routes
})

export default router
```

- 组件使用路由，src/App.vue

```
<!-- src/App.vue -->
<template>
    <div>
        <!-- 使用router-link组件来导航 -->
        <!-- 通过传入 to 属性制定链接 -->
        <!-- router-link 默认会被渲染成一个a标签 -->
        <router-link to="/home">Home</router-link>
        <router-link to="/about">About</router-link>
    </div>
    <!-- 路由占位符，路由匹配到的组件都会在这里展示 -->
    <router-view/>
</template>

<style>
    #app {
        font-family: Avenir, Helvetica, Arial, sans-serif;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
        text-align: center;
        color: #2c3e50;
        margin-top: 60px;
    }
</style>
```

- src/main.js种导入router

```
//main.js文件
import { createApp } from 'vue'
import App from './App.vue'
import Test from './components/Test.vue' //导入组件
//import axios from 'axios'
import router from './router'

const app = createApp(App)
//axios.defaults.baseURL = 'http://www.httpbin.org'
//axios.defaults.timeout = 5000
app.use(router)
app.component('Test', Test) //注册组件
app.mount('#app')
```

## 4. 路由传参

- URL传参：一般用于页面跳转，将当前数据传递到新页面，例如详情页
- **params传参**
  - 配置路由： `{path: '/user/:id', component: about}`
  - 传递方式：`<router-link to="/user/6/"></router-link>`
  - 传递后路径：`/user/6`
  - 接收参数：`$route.params.id`
- **query传参**
  - 配置路由： `{path: '/user/', component: about}`
  - 传递方式：`<router-link :to="{path:'/about', query:{id:6}}"></router-link>`
  - 传递后路径：`/user?id=6`
  - 接收参数：`$route.query.id`

## 5. 路由守卫

- 正如其名，vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。简单来说，就是在路由跳转时候的一些钩子，当从一个页面跳转到另一个页面时，可以在跳转前、中、后做一些事情。
- **每个守卫方法接收参数：**
  - to：即将要进入的目标，是一个路由对象
  - from：当前导航正要离开的路由，也是一个路由对象
  - next：可选，是一个方法，直接进入下一个路由
- 你可以使用 `router.beforeEach` 注册一个全局前置守卫，当一个导航触发时，就会异步执行这个回调函数。

```
const router = createRouter ({ ...

//添加全局前置路由守卫
router.befoEach((to,from) => {
  //这里执行具体操作
  console.log(to)
  console.log(from)
})
```

- 在网站开发中，使用导航守卫一个普遍需求：**登录验证，**即在没有登录的情况下，访问任何页面都跳转到登录页面。

- src/router/index.js

```
router.beforeEach((to, from, next) => {
    //如果用户访问登录页，直接放行
    if(to.path == '/login') {
        return next()
    }
    const token = ''  //模拟token，正常是从本地cookie或localstorage中获取
    if (token) {
        return next()  //如果有token，则正常跳转访问
    } else {
        return next('/login')  //如果没有token，跳转到登录页
    }
})
```