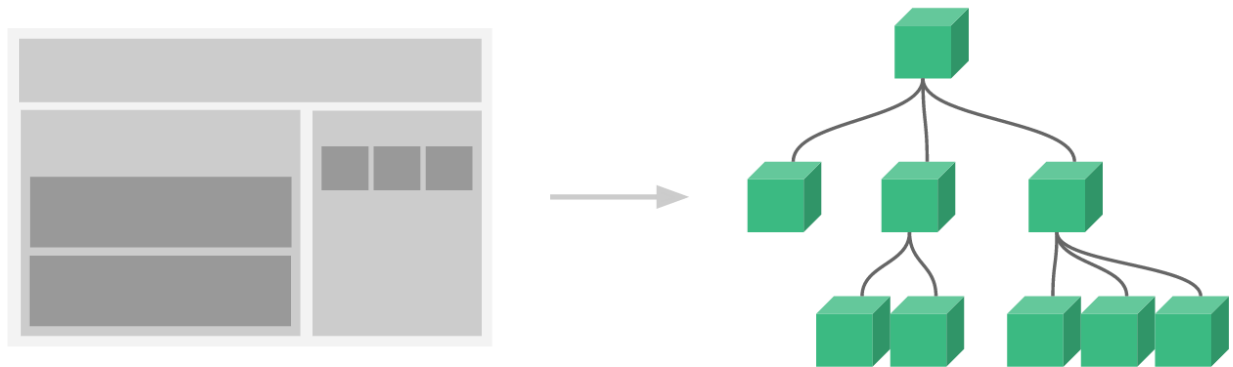


# 一、组件（重点）

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



## 1、局部组件

创建 01-1-局部组件.html

定义组件

```
1 var app = new Vue({
2   el: '#app',
3   // 定义局部组件，这里可以定义多个局部组件
4   components: {
5     //组件的名字
6     'Navbar': {
7       //组件的内容
8       template: '<ul><li>首页</li><li>学员管理</li></ul>'
9     }
10  }
11 })
```

## 使用组件

```
1 <div id="app">
2   <Navbar></Navbar>
3 </div>
```

## 2、全局组件

创建 01-2-全局组件.html

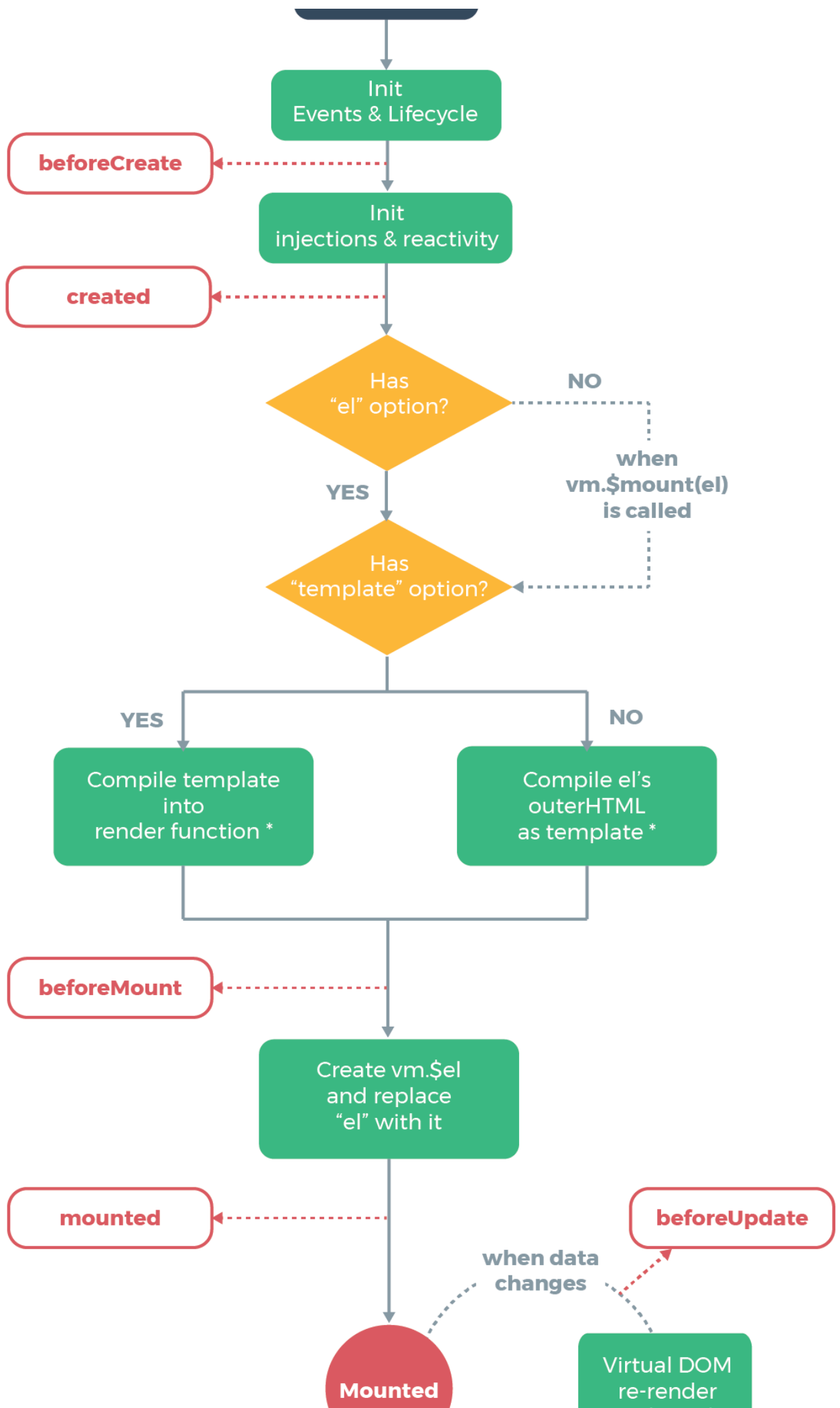
定义全局组件：components/Navbar.js

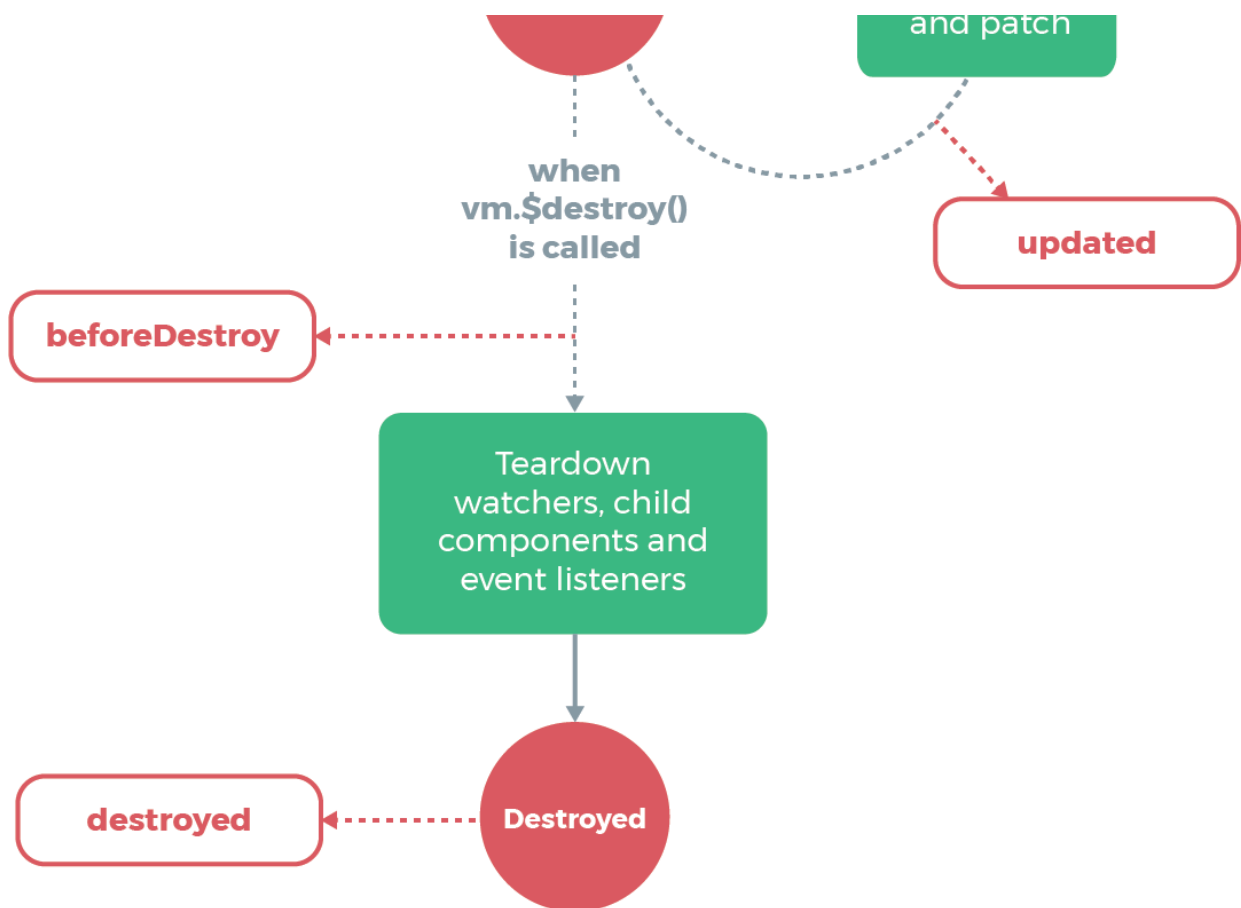
```
1 // 定义全局组件
2 Vue.component('Navbar', {
3   template: '<ul><li>首页</li><li>学员管理</li><li>讲师管理</li></ul>'
4 })
```

```
1 <div id="app">
2   <Navbar></Navbar>
3 </div>
4 <script src="vue.min.js"></script>
5 <script src="components/Navbar.js"></script>
6 <script>
7   var app = new Vue({
8     el: '#app'
9   })
10 </script>
```

## 二、实例生命周期

new Vue()





\* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

### 创建 03-vue实例的生命周期.html

```
1 data: {  
2   message: '床前明月光'  
3 },  
4 methods: {  
5   show() {  
6     console.log('执行show方法')  
7   },  
8   update() {  
9     this.message = '玻璃好上霜'  
10  }  
11 },
```

```
1 <button @click="update">update</button>
2 <h3 id="h3">{{ message }}</h3>
```

## 分析生命周期相关方法的执行时机

```
1 //===创建时的四个事件
2 beforeCreate() { // 第一个被执行的钩子方法：实例被创建出来之前执行
3     console.log(this.message) //undefined
4     this.show() //TypeError: this.show is not a function
5     // beforeCreate执行时，data 和 methods 中的数据都还没有初始化
6 },
7 created() { // 第二个被执行的钩子方法
8     console.log(this.message) //床前明月光
9     this.show() //执行show方法
10    // created执行时，data 和 methods 都已经被初始化好了！
11    // 如果要调用 methods 中的方法，或者操作 data 中的数据，最早，只能在 created 中操作
12 },
13 beforeMount() { // 第三个被执行的钩子方法
14     console.log(document.getElementById('h3').innerText) //{{ message }}
15     // beforeMount执行时，模板已经在内存中编辑完成了，尚未被渲染到页面中
16 },
17 mounted() { // 第四个被执行的钩子方法
18     console.log(document.getElementById('h3').innerText) //床前明月光
19     // 内存中的模板已经渲染到页面，用户已经可以看见内容
20 },
21
22
23 //===运行中的两个事件
24 beforeUpdate() { // 数据更新的前一刻
25     console.log('界面显示的内容：' + document.getElementById('h3').innerText)
26     console.log('data 中的 message 数据是：' + this.message)
27     // beforeUpdate执行时，内存中的数据已更新，但是页面尚未被渲染
28 },
29 updated() {
30     console.log('界面显示的内容：' + document.getElementById('h3').innerText)
31     console.log('data 中的 message 数据是：' + this.message)
32     // updated执行时，内存中的数据已更新，并且页面已经被渲染
33 }
```

## 四、路由

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）。

Vue.js 路由需要载入 vue-router 库

[创建 04-路由.html](#)

### 1、引入js

```
1 <script src="vue.min.js"></script>
2 <script src="vue-router.min.js"></script>
```

### 2、编写html

```
1 <div id="app">
2   <h1>Hello App!</h1>
3   <p>
4     <!-- 使用 router-link 组件来导航。 -->
5     <!-- 通过传入 `to` 属性指定链接。 -->
6     <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
7     <router-link to="/">首页</router-link>
8     <router-link to="/student">会员管理</router-link>
9     <router-link to="/teacher">讲师管理</router-link>
10  </p>
11  <!-- 路由出口 -->
12  <!-- 路由匹配到的组件将渲染在这里 -->
13  <router-view></router-view>
14 </div>
```

### 3、编写js

```
1 <script>
2   // 1. 定义（路由）组件。
3   // 可以从其他文件 import 进来
```

```

4   const Welcome = { template: '<div>欢迎</div>' }
5   const Student = { template: '<div>student list</div>' }
6   const Teacher = { template: '<div>teacher list</div>' }
7
8   // 2. 定义路由
9   // 每个路由应该映射一个组件。
10  const routes = [
11    { path: '/', redirect: '/welcome' }, //设置默认指向的路径
12    { path: '/welcome', component: Welcome },
13    { path: '/student', component: Student },
14    { path: '/teacher', component: Teacher }
15  ]
16
17  // 3. 创建 router 实例，然后传 `routes` 配置
18  const router = new VueRouter({
19    routes // （缩写）相当于 routes: routes
20  })
21
22  // 4. 创建和挂载根实例。
23  // 从而让整个应用都有路由功能
24  const app = new Vue({
25    el: '#app',
26    router
27  })
28
29  // 现在，应用已经启动了！
30 </script>

```

## 五、axios

axios是独立于vue的一个项目，基于promise用于浏览器和node.js的http客户端

- 在浏览器中可以帮助我们完成 ajax请求的发送
- 在node.js中可以向远程接口发送请求

### 获取数据

```
1 <script src="vue.min.js"></script>
```

```
2 <script src="axios.min.js"></script>
```

注意：测试时需要开启后端服务器，并且后端开启跨域访问权限

```
1 var app = new Vue({
2   el: '#app',
3   data: {
4     memberList: []//数组
5   },
6   created() {
7     this.getList()
8   },
9
10  methods: {
11
12    getList(id) {
13      //vm = this
14      axios.get('http://localhost:8081/admin/ucenter/member')
15        .then(response => {
16          console.log(response)
17          this.memberList = response.data.data.items
18        })
19        .catch(error => {
20          console.log(error)
21        })
22    }
23  }
24 })
```

控制台查看输出

## 2、显示数据

```
1 <div id="app">
2   <table border="1">
3     <tr>
4       <td>id</td>
5       <td>姓名</td>
6     </tr>
```



```
7      <tr v-for="item in memberList">
8          <td>{{item.memberId}}</td>
9          <td>{{item.nickname}}</td>
10     </td>
11 </tr>
12 </table>
13 </div>
```

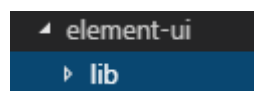
## 六、element-ui :

element-ui 是饿了么前端出品的基于 Vue.js的 后台组件库，方便程序员进行页面快速布局和构建

官网：<http://element-cn.eleme.io/#/zh-CN>

创建 06-element-ui.html

将element-ui引入到项目



### 1、引入css

```
1 <!-- import CSS -->
2 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
```

### 2、引入js

```
1 <!-- import Vue before Element -->
2 <script src="vue.min.js"></script>
3 <!-- import JavaScript -->
4 <script src="element-ui/lib/index.js"></script>
```

### 3、编写html

```
1 <div id="app">
2   <el-button @click="visible = true">Button</el-button>
3   <el-dialog :visible.sync="visible" title="Hello world">
4     <p>Try Element</p>
5   </el-dialog>
6 </div>
```

关于.sync的扩展阅读

<https://www.jianshu.com/p/d42c508ea9de>

## 4、编写js

```
1 <script>
2   new Vue({
3     el: '#app',
4     data: function () { //定义Vue中data的另一种方式
5       return { visible: false }
6     }
7   })
8 </script>
```

**测试**

其他ui组件我们在项目中学习