第十六章：es6&Vue

第1节：es6

1.1 简介

> ECMAScript 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准，已经在 2015 年 6 月正式发布了。它的目标，是使得 JavaScript 语言可以用来编写复杂的大型应用程序，成为企业级开发语言。
> ① ECMAScript 和 JavaScript 的关系
> 1996 年 11 月，JavaScript 的创造者 Netscape 公司，决定将 JavaScript 提交给标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。
> 因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（另外的 ECMAScript 方言还有 Jscript 和 ActionScript）。
> ② ES6 与 ECMAScript 2015 的关系
> 2011 年，ECMAScript 5.1 版发布后，就开始制定 6.0 版了。因此，ES6 这个词的原意，就是指 JavaScript 语言的下一个版本。
> ES6 的第一个版本，在 2015 年 6 月发布，正式名称是《ECMAScript 2015 标准》（简称 ES2015）。
> 2016 年 6 月，小幅修订的《ECMAScript 2016 标准》（简称 ES2016）如期发布，这个版本可以看作是 ES6.1 版，因为两者的差异非常小，基本上是同一个标准。根据计划，2017 年 6 月发布 ES2017 标准。
> 因此，ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等，而 ES2015 则是正式名称，特指该年发布的正式版本的语言标准。本书中提到 ES6 的地方，一般是指 ES2015 标准，但有时也是泛指“下一代 JavaScript 语言”。

1.2 基本语法

> ES标准中不包含 DOM 和 BOM的定义，只涵盖基本数据类型、关键字、语句、运算符、内建对象、内建函数等通用语法。

1.2.1 let声明变量

```
// var 声明的变量没有局部作用域
// let 声明的变量  有局部作用域
{
var a = 0
let b = 1
}
console.log(a)  // 0
console.log(b)  // ReferenceError: b is not defined

// var 可以声明多次
// let 只能声明一次
```

```
var m = 1
var m = 2
let n = 3
let n = 4
console.log(m)  // 2
console.log(n)  // Identifier 'n' has already been declared

// var 会变量提升
// let 不存在变量提升
console.log(x)  //undefined
var x = "apple"
console.log(y)  //ReferenceError: y is not defined
let y = "banana"
```

1.2.2 const声明常量

```
// 1、声明之后不允许改变
const PI = "3.1415926"
PI = 3  // TypeError: Assignment to constant variable.

// 2、一但声明必须初始化，否则会报错
const MY_AGE  // SyntaxError: Missing initializer in const declaration
```

1.2.3 解构赋值

解构赋值是对赋值运算符的扩展。
他是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。
在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

```
//1、数组解构
// 传统
let a = 1, b = 2, c = 3
console.log(a, b, c)
// ES6
let [x, y, z] = [1, 2, 3]
console.log(x, y, z)

//2、对象解构
let user = {name: 'Helen', age: 18}
// 传统
let name1 = user.name
let age1 = user.age
console.log(name1, age1)
// ES6
```

```
let { name, age } =  user//注意：解构的变量必须是user中的属性
console.log(name, age)
```

1.2.4 模板字符串

模板字符串相当于加强版的字符串，用反引号 `,除了作为普通字符串，还可以用来定义多行字符串，还可以在字符串中加入变量和表达式。

```
// 1、多行字符串
let string1 =  `Hey,
can you stop angry now?`
console.log(string1)
// Hey,
// can you stop angry now?

// 2、字符串插入变量和表达式。变量名写在 ${} 中，${} 中可以放入 JavaScript 表达式。
let name = "Mike"
let age = 27
let info = `My Name is ${name},I am ${age+1} years old next year.`
console.log(info)
// My Name is Mike,I am 28 years old next year.

// 3、字符串中调用函数
function f(){
   return "have fun!"
}
let string2 = `Game start,${f()}`
console.log(string2);  // Game start,have fun!
```

1.2.5 声明对象简写

```
const age = 12
const name = "Amy"
// 传统
const person1 = {age: age, name: name}
console.log(person1)
// ES6
const person2 = {age, name}
console.log(person2) //{age: 12, name: "Amy"}
```

1.2.6 定义方法简写

```
// 传统
const person1 = {
   sayHi:function(){
       console.log("Hi")
   }
}
person1.sayHi();//"Hi"
// ES6
const person2 = {
   sayHi(){
       console.log("Hi")
   }
}
person2.sayHi()  //"Hi"
```

1.2.7 对象拓展运算符

拓展运算符（...）用于取出参数对象所有可遍历属性然后拷贝到当前对象。

```
// 1、拷贝对象
let person1 = {name: "Amy", age: 15}
let someone = { ...person1 }
console.log(someone)  //{name: "Amy", age: 15}

// 2、合并对象
let age = {age: 15}
let name = {name: "Amy"}
let person2 = {...age, ...name}
console.log(person2)  //{age: 15, name: "Amy"}
```

1.2.8 函数的默认参数

```
function showInfo(name, age = 17) {
   console.log(name + "," + age)
}
// 只有在未传递参数，或者参数为 undefined 时，才会使用默认参数
// null 值被认为是有效的值传递。
showInfo("Amy", 18)  // Amy,18
showInfo("Amy", "")  // Amy,
showInfo("Amy", null)  // Amy, null
showInfo("Amy")     // Amy,17
showInfo("Amy", undefined) // Amy,17
```

1.2.9 不定参数

不定参数用来表示不确定参数个数，形如，...变量名，由...加上一个具名参数标识符组成。具名参数只能放在参数列表的最后，并且有且只有一个不定参数。

```
function f(...values) {
   console.log(values.length)
}
f(1, 2)      //2
f(1, 2, 3, 4)  //4
```

1.2.10 箭头函数

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：参数 => 函数体

```
// 传统
var f1 = function(a){
   return a
}
console.log(f1(1))
// ES6
var f2 = a => a
console.log(f2(1))

// 当箭头函数没有参数或者有多个参数，要用 () 括起来。
// 当箭头函数函数体有多行语句，用 {} 包裹起来，表示代码块，
// 当只有一行语句，并且需要返回结果时，可以省略 {} ，结果会自动返回。
var f3 = (a,b) => {
   let result = a+b
   return result
}
```

```
console.log(f3(6,2))  // 8
// 前面代码相当于:
var f4 = (a,b) => a+b
```

第2节：Vue

2.1 介绍

Vue.js（读音 /vjuː/，类似于 view） 是一套构建用户界面的渐进式框架。Vue 只关注视图层， 采用自底向上增量开发的设计。
Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。
Vue.JS是优秀的前端 JavaScript 框架

2.2 作用

随着项目业务场景的复杂,传统模式(html+jquery)已无法满足需求,就出现了Angular/React/Vue等框架
企业需求、主流框架之一、易用、灵活、高效
最大程度上解放了 DOM 操作
单页web项目开发
传统网站开发

2.3 核心特征

① 解耦视图与数据
② M-V-VM模型 关注模型和视图
M：即Model，模型，包括数据和一些基本操作。
V：即View，视图，页面渲染结果
VM：即View-Model，模型和视图间的双向操作
③ 双向数据绑定

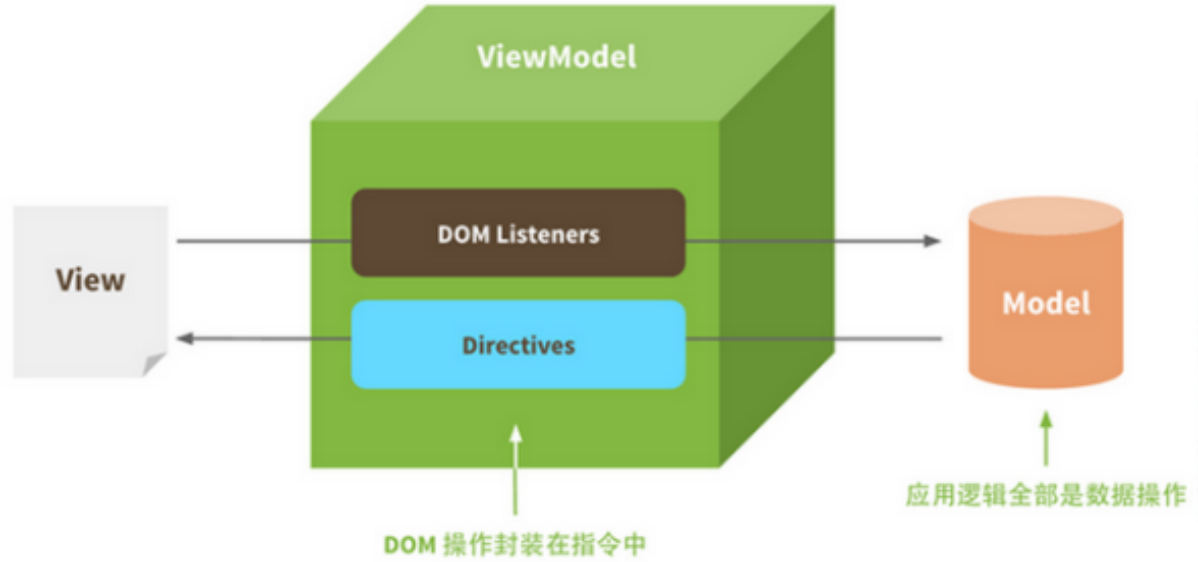
2.3.1 MVVM之前

开发人员从后端获取需要的数据模型，然后要通过DOM操作Model渲染到View中。而后当用户操作视图，我们还需要通过DOM获取View中的数据，然后同步到Model中

2.3.2 MVVM之后

而MVVM中的VM要做的事情就是把DOM操作完全封装起来，开发人员不用再关心Model和View之间是如何互相影响的：
- 只要我们Model发生了改变，View上自然就会表现出来。
- 当用户修改了View，Model中的数据也会跟着改变

把开发人员从繁琐的DOM操作中解放出来，把关注点放在如何操作Model上



2.4 使用

2.4.1 下载安装

vue是一个前端框架，也是其实是一个js文件，下载vue.js文件并在页面中引入
vue.js的下载方式：
① 可以引入在线的vue.js(公共的CDN服务)

```
<!-- 开发环境版本，包含了用帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
或
<!-- 生产环境版本，优化了尺寸和速度 -->
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
```

② 可以离线下载vue.js
开发版本：https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js
生产版本：https://cdn.jsdelivr.net/npm/vue@2
③ npm包资源管理器，可以下载vue.js
初始化：npm init -y
安装vue：npm install vue --save
注：切记重启计算机

注意：
NPM是Node提供的模块管理工具，可以非常方便的下载安装很多前端框架，包括Jquery、AngularJS、VueJs都有。为了后面学习方便，我们先安装node及NPM工具
node.js下载地址：https://nodejs.org/en/download/，安装完成Node应该自带了NPM了，在控制台输入npm -v查看
注：
① 在v12.16.2以上版本就不在支持window7系统。
② npm默认的仓库地址是在国外网站，速度较慢，建议大家设置到淘宝镜像。但是切换镜像是比较麻烦的。推荐一款切换镜像的工具：nrm
安装命令：npm install nrm -g 这里-g代表全局安装
查看npm的仓库列表：nrm ls

```
指定镜像源：nrm use taobao
测试速度：nrm test npm
```

2.4.2 实现

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
  <div id="app">
      <h2>{{name}}, 欢迎你! </h2>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue"></script>
  <script type="text/javascript">
      //生成一个Vue实例
      var app=new Vue({
          el:"#app",//el ,即是element。要渲染的页面元素
          data:{//数据
              name:"优就业"
          }
      })
  </script>
</body>
</html>

Vue参数详解:
  1. body中,设置Vue管理的视图<div id="app"></div>
  2. 引入vue.js
  3. 实例化Vue对象 new Vue();
  4. 设置Vue实例的选项:如el、data...
    new Vue({选项:值});
  5. 在<div id='app'></div>中通过{{ }}使用data中的数据
```

2.5 常见指令

指令 (Directives) 是带有 v- 前缀的特殊attribute。是Vue框架提供的语法，扩展了html标签的功能、大部分的指令的值是js的表达式。用于取代DOM操作

2.5.1 `v-text` 和 `v-html`

> 类似innerText和innerHTML
> ① v-text：更新标签中的内容
> ② v-html：更新标签中的内容/标签
> 案例：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
     <p v-text="t">这是啥</p>
     <p v-html="h">你猜</p>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data:{
             t:"<i>哈哈</i>",
             h:"<i>嘿嘿</i>"
         }
     })
 </script>
</body>
</html>
```

2.5.2 `v-if` 和 `v-show`

> 根据表达式的boolean值进行判断是否渲染该元素
> 案例：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
     <p v-if="b">这是啥</p>
     <p v-show="b">你猜</p>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
```

```
    const v = new Vue({
        el:"#app",
        data:{
           b:false
        }
    })
 </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▼<div id="app">
      <!---->
      <p style="display: none;">你猜</p>
    </div>
    <script src="./node_modules/vue/dist/vue.js"></script>
  ▼<script type="text/javascript">
            const v = new Vue({
                el:"#app",
                data:{
                   b:false
                }
            })

    </script>
  </body>
</html> == $0
```

2.5.3 v-on

```
① 作用：使用 v-on 指令绑定 DOM 事件，并在事件被触发时执行一些 JavaScript 代码。
② 语法：
   v-on:事件名.修饰符 = "methods中的方法名";
   v-on的简写方法：@事件名.修饰符 = "methods中的方法名";
③ 案例：
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
     <button v-on:click="num++">按钮01</button><br />{{num}}
     <hr>
     <button @click="fn1()">按钮02</button>
     <button @click="fn2(num)">按钮03</button>
```

```
    <button @click="fn3">按钮04</button>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data:{
            num:0
         },
         methods:{
             fn1:function(){
                 console.info("fn1 调用了。。。"+this.num);
                 this.num++;
             },
             fn2:function(n){
                 console.info("fn2 调用了...."+n);
             },
             fn3:function(){
                 console.info("fn3 调用了....");
             }
         }
     })
 </script>
</body>
</html>
```

2.5.4 `v-for`

```
① 作用：列表渲染,当遇到相似的标签结构时,就用v-for去渲染
② 格式：
    【1】(item,index) in 数组或集合
    参数item:数组中的每个元素
    参数index:数组中元素的下标
    【2】(value, key, index) in 对象
    参数index:对象中每对key-value的索引 从0开始
    参数key:键
    参数value:值
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
    <!-- 遍历数组 -->
     <table border="1">
         <!-- <tr v-for="item in userList"></tr> -->
```

```
        <tr v-for="(item, index) in userList">
            <td>{{index}}</td>
            <td>{{item.id}}</td>
            <td>{{item.username}}</td>
            <td>{{item.age}}</td>
        </tr>
    </table>
   <!-- 遍历对象 -->
      <!-- 直接访问 -->
      <form action="">
          <p><label>id: <input type="text" v-model="user.id"></label></p>
          <p><label>用户名: <input type="text" v-model="user.username"></label></p>
          <p><label>年龄: <input type="text" v-model="user.age"></label></p>
      </form>
      <!-- 遍历 -->
      <form action="">
          <!-- <p v-for="value in user"> -->
          <p v-for="(value, key, index) in user">
              <label>{{index}}-{{key}}: <input type="text" v-model="user[key]">
</label>
          </p>
      </form>

 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data: {
             userList: [
                 { id: 1, username: 'helen', age: 18 },
                 { id: 2, username: 'peter', age: 28 },
                 { id: 3, username: 'andy', age: 38 }
             ],
            user:{
                 id: 1,
                 username: 'helen',
                 age: 18
             }
         }
     })
 </script>
</body>
</html>
```

2.5.5 v-bind

① 作用：可以绑定标签上的任何属性
② 格式：v-bind:属性="值"
③ 简写格式：:属性="值"
④ 属性值一部分进行替换的格式：:属性="'常量值' + vue对象data中的数据"
⑤ 案例：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
     <font v-bind:color="v1">有就业</font>
     <font :color="v2">这是v-bind</font>
     <a :href="'http://'+u">百度</a>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data:{
             v1:"red",
             v2:"yellow",
             u:"www.baidu.com"
         }
     })
 </script>
</body>
</html>
```

2.5.6 v-model

① 作用：表单元素的绑定
② 特点：双向数据绑定
   （1）vue对象中的数据发生变化可以更新到界面
   （2）通过界面可以更改vue对象中数据
   （3）v-model 会忽略所有表单元素的 value、 checked 、 selected 特性的初始值而总是将 Vue 实例的数据作为数据来源。应该在data选项中声明初始值。
③ 案例：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
```

```
</head>
<body>
 <div id="app">
    <form>
        用户名: <input type="text" v-model="user.name"><br>
        用户名: <input type="text" v-model="v"><br>
        密码: <input type="password" v-model="user.password"><br>
        <input type="button" @click="fun1" value="获取">
        <input type="button" @click="fun2" value="修改">
    </form>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data:{
             user:{name:"root",password:"1234"},
             v:"hehe"
         },
         methods:{
             fun1:function(){
                 console.info(this.user.name);
                 console.info(this.user.password);
             },
             fun2:function(){
                 this.user.name="admin";
                 this.user.password="111";
             }
         }
     })
 </script>
</body>
</html>
```

2.5.7 计算属性

在插值表达式中使用js表达式是非常方便的，而且也经常被用到。
但是如果表达式的内容很长，就会显得不够优雅，而且后期维护起来也不方便
如下:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
    {{message}}
```

```
    <p>{{birthday}}</p>
    没计算属性：<p>{{new Date(birthday).getFullYear() + '-'+ new
Date(birthday).getMonth()+1+ '-' + new Date(birthday).getDate()}}</p>
    计算属性：<p>{{getBirth}}</p>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
    const v = new Vue({
        el:"#app",
        data:{
            message:"",
            birthday:1610669793429
        },
        created(){
            this.message = "创建啦.....";
            console.info(this);
        },
        computed:{
            getBirth(){
             const d = new Date(this.birthday);
             return d.getFullYear() + "-" + d.getMonth()+1 + "-" + d.getDate();
            }
        }
    })
 </script>
</body>
</html>
```

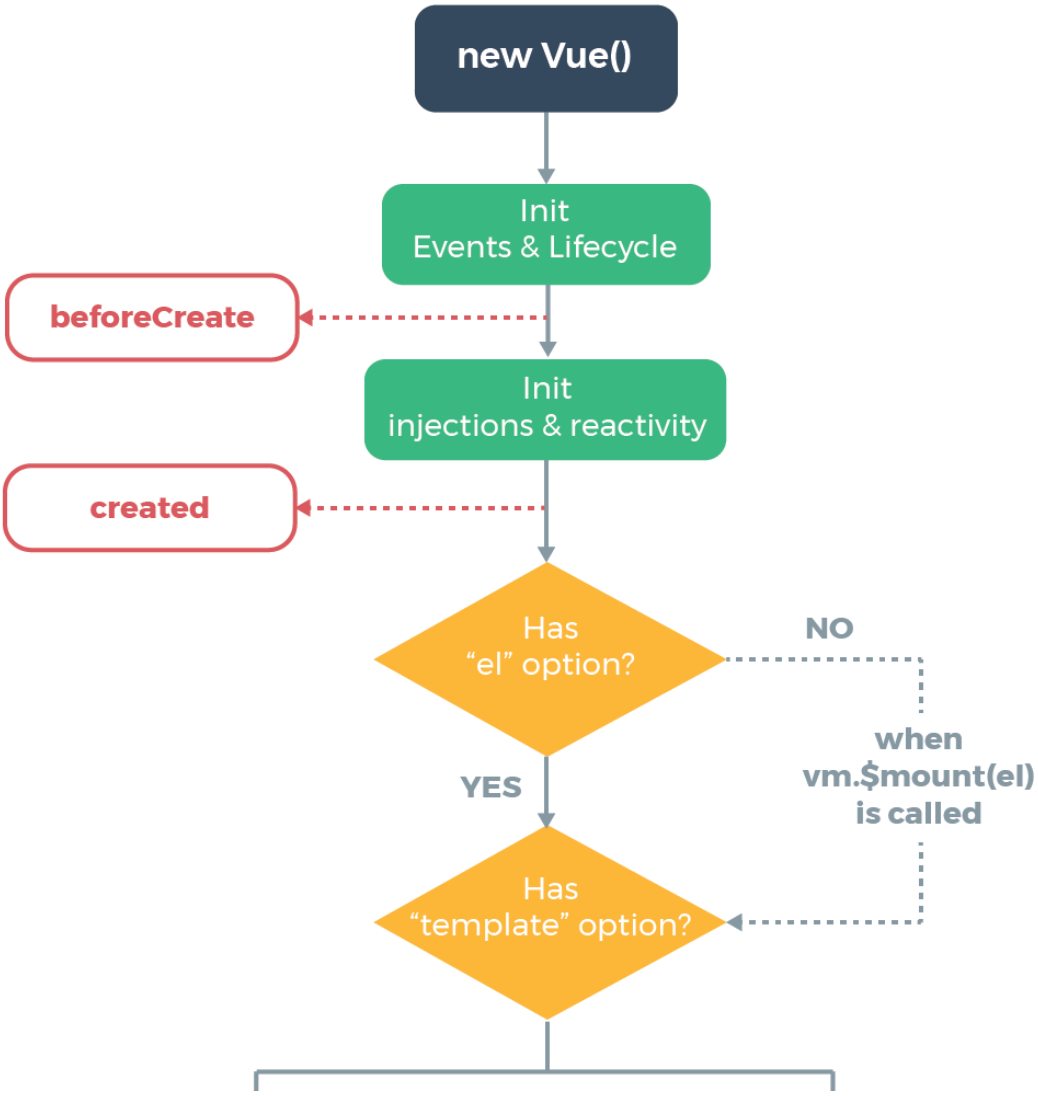
2.5.8 watch

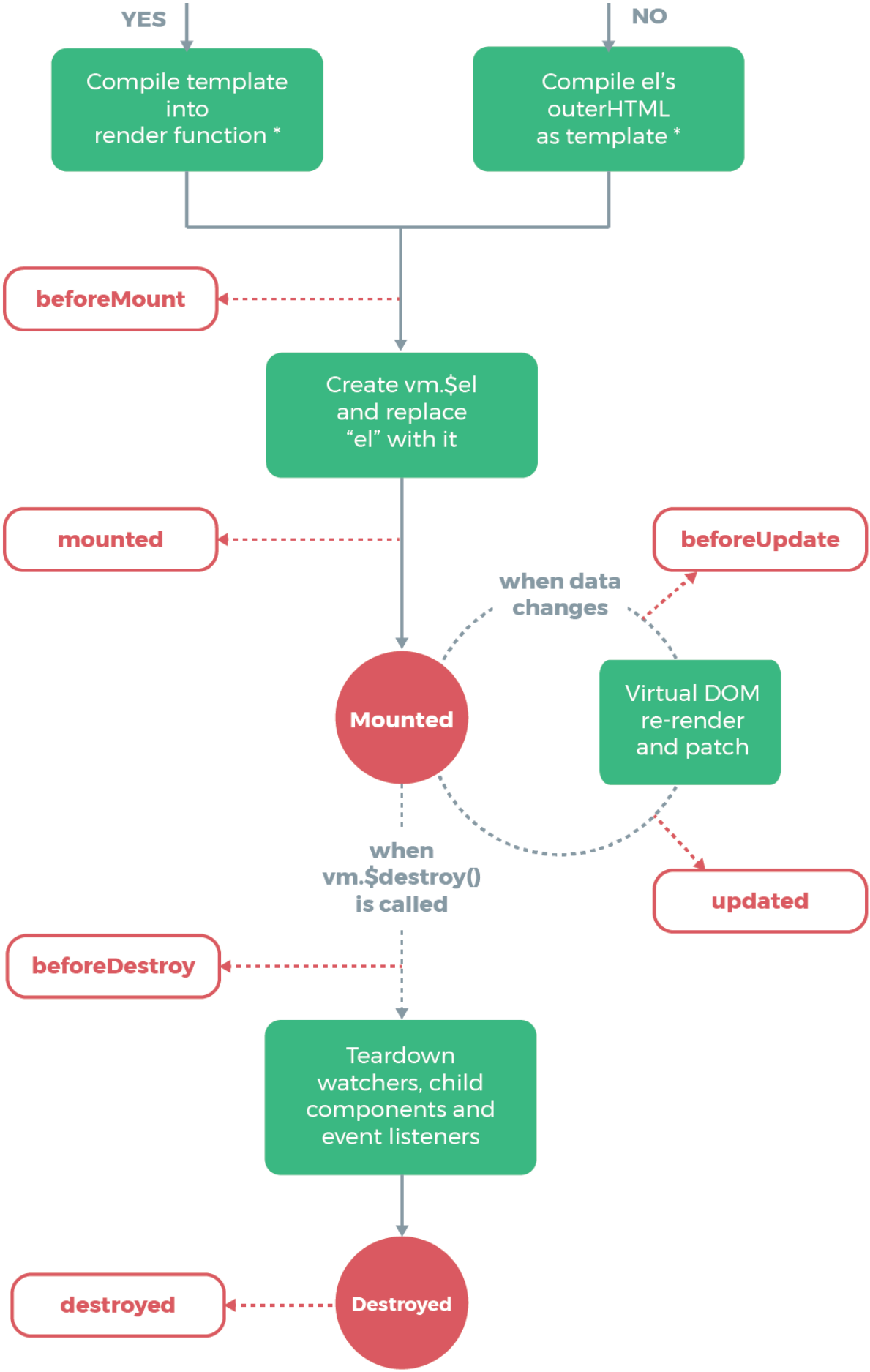
```
watch可以让我们监控一个值的变化。从而做出相应的反应。
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
    <input type="text" v-model="hello">
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     const v = new Vue({
         el:"#app",
         data:{
             hello:""
         },
```

```
        watch:{
            hello(newVal, oldVal){
                console.log(newVal, oldVal);
            }
        }
    })
 </script>
</body>
</html>
查看控制台
```

2.6 生命周期

每个 Vue 实例在被创建时都要经过一系列的初始化过程 ： 创建实例，装载模板，渲染模板等等。Vue为生命周期中的每个状态都设置了钩子函数（监听函数）。每当Vue实例处于不同的生命周期时，对应的函数就会被触发调用。




YES
NO
Compile template into render function *
Compile el's outerHTML as template *
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

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

钩子函数：
如：created代表在vue实例创建后；我们可以在Vue中定义一个created函数，代表这个时期的构造函数：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
    {{message}}
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
    const v = new Vue({
        el:"#app",
        data:{
            message:""
        },
        created(){
            this.message = "创建啦.....";
        }
    })
 </script>
</body>
</html>
```

2.7 组件

2.7.1 定义全局组件

我们通过Vue的component方法来定义一个全局组件。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
```

```
    <!--使用定义好的组件-->
    <con></con>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script type="text/javascript">
    // 参数1：组件名 参数2：组件参数
    Vue.component("con",{
        template:"<button @click='count++'>点我啊--{{count}}</button>",
        data(){
            return {
                count:0
            }
        }
    })
    const v = new Vue({
        el:"#app"
    })
</script>
</body>
</html>
```

特点：

- 组件其实也是一个Vue实例，因此它在定义时也会接收：data、methods、生命周期函数等
- 不同的是组件不会与页面的元素绑定，否则就无法复用了，因此没有el属性。
- 但是组件渲染需要html模板，所以增加了template属性，值就是HTML模板
- 全局组件定义完毕，任何vue实例都可以直接在HTML中通过组件名称来使用组件了。
- data的定义方式比较特殊，必须是一个函数。

注：定义组件要在Vue对象之前声明

2.7.2 定义局部组件

一旦全局注册，就意味着即便以后你不再使用这个组件，它依然会随着Vue的加载而加载。因此，对于一些并不频繁使用的组件，我们会采用局部注册。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Vue入门</title>
</head>
<body>
 <div id="app">
     <conn></conn>
 </div>
 <script src="./node_modules/vue/dist/vue.js"></script>
 <script type="text/javascript">
     // 局部组件
     const conn = {
```

```
        template:"<button @click='count1++'>点我干啥 都{{count1}}次了</button>",
        data(){
            return {
                count1:0
            }
        }
    }
    const v = new Vue({
        el:"#app",
        components:{
            conn:conn //
        }
    })

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

- components就是当前vue对象子组件集合。
  - 其key就是子组件名称
  - 其值就是组件对象的属性
- 效果与刚才的全局注册是类似的，不同的是，这个conn组件只能在当前的Vue实例中使用

注：定义组件要在Vue对象之前声明

2.8 Vue的Ajax(axios)

在Vue.js中发送网络请求本质还是ajax，我们可以使用插件方便操作。
1. vue-resource：Vue.js的插件，已经不维护，不推荐使用
2. axios ：不是vue的插件，可以在任何地方使用，推荐
3. 通过Http请求的不同类型(POST/DELETE/PUT/GET)来判断是什么业务操作(CRUD) HTTP方法规则举例

说明：
  ① POST Create 新增一个没有id的资源
  ② GET Read 取得一个资源
  ③ PUT Update 更新一个资源。或新增一个含 id 资源(如果 id 不存在)
  ④ DELETE Delete 删除一个资源

2.8.1 安装

方式1：使用npm安装
  命令：npm install axios

方式2：使用cdn链接axios

```
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

2.8.2 axios请求

```
axios({
  // 请求方式
 method: 'post',
 url: 'api',
 // 传递参数
 data: obj, // URLSearchParam()
 // 设置请求头信息
 headers: {
   key: value
 },
 responseType: 'json'
}).then(response => {
 // 请求成功
 let res = response.data;
 console.log(res);
}).catch(error => {
 // 请求失败,
 console.log(error);
});
```

2.8.3 GET请求

```
axios.get('/user?id=12345')
 .then(response => {
     console.log(response.data);
 })
 .catch(error => {
     console.dir(error)
});
```

2.8.4 POST请求

```
axios.post('/user', "URLSearchParams") .then(response => {
     console.log(response.data);
 })
 .catch(error => {
     console.dir(err)
});
```

补充：

为方便起见，为所有支持的请求方法提供了别名

```
axios.request(configfig)
axios.get(url[, configfig])
axios.delete(url[, configfig])
axios.head(url[, configfig])
axios.post(url[, data[, configfig]])
axios.put(url[, data[, configfig]])
axios.patch(url[, data[, configfig]])
```

2.8.5 跨域问题

什么是跨域?

指的是浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是浏览器对javascript施加的安全限制。

什么是同源策略?

是指协议，域名，端口都要相同，其中有一个不同都会产生跨域，在请求数据时，浏览器会在控制台中报一个异常，提示拒绝访问。

跨域问题怎么出现的?

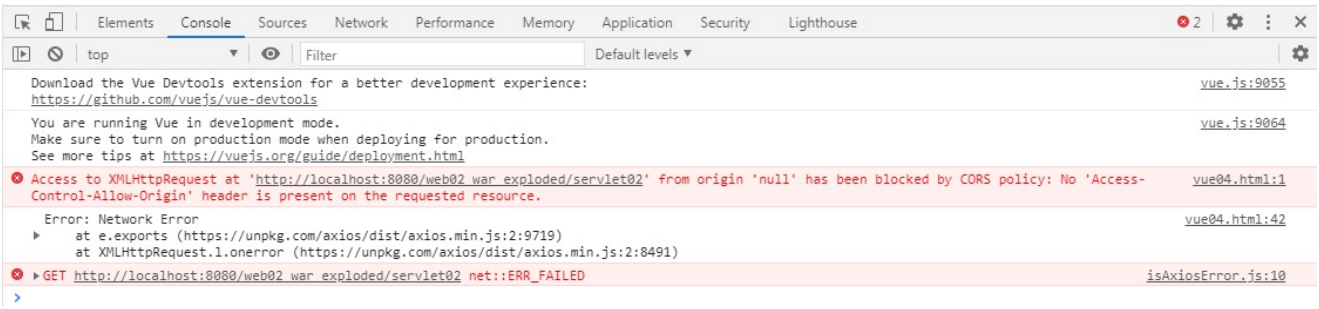
开发一些前后端分离的项目，比如使用 Servlet + Vue 开发时，后台代码在一台服务器上启动，前台代码在另外一台电脑上启动，此时就会出现问题。

比如：

后台 地址为 http://192.168.70.77:8081

前台 地址为 http://192.168.70.88:8080

此时 ip 与 端口号不一致， 不符合同源策略，造成跨域问题。



如何解决：

方式1：后台解决(自定义过滤器)

```
 HttpServletResponse response = (HttpServletResponse) resp;
HttpServletRequest request = (HttpServletRequest) req;

// 不使用*，自动适配跨域域名，避免携带Cookie时失效
String origin = request.getHeader("Origin");
response.setHeader("Access-Control-Allow-Origin", origin);

// 自适应所有自定义头
String headers = request.getHeader("Access-Control-Request-Headers");
```

```
        response.setHeader("Access-Control-Allow-Headers", headers);
        response.setHeader("Access-Control-Expose-Headers", headers);

        // 允许跨域的请求方法类型
        response.setHeader("Access-Control-Allow-Methods", "*");
        // 预检命令（OPTIONS）缓存时间，单位：秒
        response.setHeader("Access-Control-Max-Age", "3600");
        // 明确许可客户端发送Cookie，不允许删除字段即可
        response.setHeader("Access-Control-Allow-Credentials", "true");

        chain.doFilter(request, response);
```

方式2：前端解决

① 首先npm安装好axios，其次在main.js中引入

```
import axios from 'axios'
//把axios挂载到vue的原型中，在vue中每个组件都可以使用axios发送请求
Vue.prototype.$axios = axios
 //重要在于这里，Vue.prototype.HOME = '/api'是一个定值，默认指向localhost，所有修改指向路径为'/api'，配置文件index.js定义的可跨域路径
Vue.prototype.HOME = '/api'
```

② 修改上述所说的config>index.js配置文件

```
module.exports = {
      dev: {
        // Paths
        assetsSubDirectory: 'static',
        assetsPublicPath: '/',
        proxyTable: {      //axios跨域处理
          '/api': {        //此处并非和url一致
            target:'http://192.168.2.80:8081/',
            changeOrigin:true, //允许跨域
            pathRewrite:{
              '^/api': ''
            }
          }
        }
    }
```

③ 在文件中发送axios

```
<template>
    <div id="app">
      <button @click="fn">点击发送axios</button>
      <router-view/>
    </div>
  </template>

  <script>
      export default {
        name: 'App',
        methods:{
          fn:function(){
```

```
                this.$axios.get(this.HOME+'/web02_war_exploded/servlet02')
                    .then(response => {
                        console.log(response.data);
                    })
                    .catch(error => {
                        console.dir(error)
                });
            }
        }
    }
    </script>
```

2.8.6 案例

后台Servlet

```
package com.ujiuye.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "AxiosTestServlet",urlPatterns = "/axiosTest")
public class AxiosTestServlet extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     // 1. 获取请求参数
     String name = request.getParameter("name");
     String age = request.getParameter("age");

     System.out.println(name+"\t"+age);

     // 2. 响应数据
     String json = "{\"id\":1001,\"name\":\"zhang3\"}";
     PrintWriter writer = response.getWriter();

     writer.print(json);
 }

 protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     doPost(request, response)
 }
}
```

前台代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <div id="app">
     <button @click="sendPostAxios">点击发送Post请求</button>

     <form action="" method="" >
         <p v-for="(value, key, index) in xx" :key="index">
             <label>{{key}}: </label>: <input type="text" v-model="xx[key]">
         </p>
     </form>
 </div>
 <script src="../Vue_01/vue.js"></script>
 <script src="../Vue_04/axios.min.js"></script>
 <script>
     new Vue({
         el:"#app",
         data:{
             xx:{

             }
         },
         methods:{
             sendPostAxios(){
                 // 方式3：发送axios

axios.post("http://localhost:8080/axiosTest_Web_exploded/axiosTest","name=xx&age=18").then(response => {
                     this.xx = response.data
                     console.log(this.xx)
                 }).catch(error => {
                     console.log(error)
                 })
             }
         }
     })
 </script>
</body>
</html>
```

> 注：别忘了配置跨域过滤器哦