1. Vue概述

Vue：渐进式JavaScript框架
声明式渲染→组件系统→客户端路由→集中式状态管理→项目构建

1. 易用：熟悉HTML、CSS、JavaScript知识后，可快速上手Vue
2. 灵活：在一个库和一套完整框架之间自如伸缩
3. 高效：20kB运行大小，超快虚拟DOM

2. Vue基本使用

2.1 传统开发模式对比

1. 原生JS

```
<div id="msg"></div>
<script type="text/javascript">
    var msg = 'Hello World';
    var div = document.getElementById('msg');
    div.innerHTML = msg;
</script>
```

2. jQuery

```
<div id="msg"></div>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
    var msg = 'Hello World';
    $('#msg').html(msg);
</script>
```

2.2 Vue.js之HelloWorld基本步骤

```
<div id="app">
<div>{{msg}}</div>
</div>
<script type="text/javascript" src="js/vue.js"></script>
<script type="text/javascript">
    new Vue({
        el: '#app',
        data: {
        msg: 'HelloWorld'
        }
    });
</script>
```

2.3 Vue.js之HelloWorld细节分析

1. 实例参数分析
    1. el： 元素的挂载位置（值可以是CSS选择器或者DOM元素）
    2. data：模型数据（值是一个对象）
2. 插值表达式用法
    1. 将数据填充到HTML标签中
    2. 插值表达式支持基本的计算操作
3. Vue代码运行原理分析
    1. 概述编译过程的概念（Vue语法→原生语法）

2.4 Vue.js之HelloWorld实例

```
<!DOCTYPE html>
<html lang="en">

<head>
```

```
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <div id="app">
        <div>{{msg}}</div>
        <div>{{1 + 2}}</div>
        <div>{{msg + '----' + 123}}</div>
    </div>
    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        /*
              Vue的基本使用步骤
              1、需要提供标签用于填充数据
              2、引入vue.js库文件
              3、可以使用vue的语法做功能了
              4、把vue提供的数据填充到标签里面
            */
        var vm = new Vue({
            el: '#app',
            data: {
                msg: 'Hello Vue'
            }
        });
    </script>
</body>

</html>
```

3. Vue模板语法

3.1 模板语法概述

1. 如何理解前端渲染?

把数据填充到HTML标签中

2. 前端渲染方式

1. 原生js拼接字符串
2. 使用前端模板引擎
3. 使用vue特有的模板语法

3. 原生js拼接字符串

基本上就是将数据以字符串的方式拼接到HTML标 签中，前端代码风格大体上如下。

```
var d = data.weather;
var info = document.getElementById('info');
info.innerHTML = '';
for(var i=0;i<d.length;i++){
    var date = d[i].date;
    var day = d[i].info.day;
    var night = d[i].info.night;
    var tag = ''; tag += '<span>日期：'+date+'</sapn><ul>'; tag += '<li>白天天
气：'+day[1]+'</li>' tag += '<li>白天温度：'+day[2]+'</li>' tag += '<li>白天风
向：'+day[3]+'</li>' tag += '<li>白天风速：'+day[4]+'</li>' tag += '</ul>';
    var div = document.createElement('div');
    div.innerHTML = tag;
    info.appendChild(div);
}
```

缺点：不同开发人员的代码风格差别很大，随着业 务的复杂，后期的维护变得逐渐困难起来。

4. 使用前端模板引擎

如下代码是基于模板引擎art-template的一段代 码，与拼接字符串相比，代码明显规范了很多， 它拥有自己的一套模板语法规则。

```
<script id="abc" type="text/html">
    {{if isAdmin}}
    <h1>{{title}}</h1>
    <ul>
        {{each list as value i}} <li>索引 {{i + 1}} ：{{value}}</li> {{/each}}
    </ul>
    {{/if}}
</script>
```

优点：大家都遵循同样的规则写代码，代码可读性 明显提高了，方便后期的维护。
缺点：没有专门提供事件机制。

5. 模板语法概览

1. 差值表达式
2. 指令
3. 事件绑定
4. 属性绑定
5. 样式绑定
6. 分支循环结构

3.2 指令

1. 什么是指令?

1. 什么是自定义属性
2. 指令的本质就是自定义属性
3. 指令的格式：以v-开始（比如：v-cloak）

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style type="text/css">
        /*
    1、通过属性选择器 选择到 带有属性 v-cloak的标签
       让他隐藏
 */

        [v-cloak] {
            display: none;
        }
    </style>
</head>

<body>
    <div id="app">
        <!-- 2、 让带有插值 语法的    添加 v-cloak 属性
         在 数据渲染完场之后，v-cloak 属性会被自动去除，
         也就是对应的标签会变为可见
    -->
        <div v-cloak>{{msg}}</div>

    </div>

    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        var vm = new Vue({
            el: '#app',
            data: {
                msg: 'Hello Vue'
            }
        });
    </script>
</body>

</html>
```

2. v-cloak指令用法

1. 插值表达式存在的问题：“闪动”
2. 如何解决该问题：使用v-cloak指令
3. 解决该问题的原理：先隐藏，替换好值之后再显示最终的值

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">
  [v-cloak]{
    display: none;
  }
  </style>
```

```
</head>
<body>
  <div id="app">
    <div v-cloak>{{msg}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      v-cloak指令的用法
      1、提供样式
        [v-cloak]{
          display: none;
        }
      2、在插值表达式所在的标签中添加v-cloak指令

      背后的原理：先通过样式隐藏内容，然后在内存中进行值的替换，替换好之后再显示最终的结果
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Hello Vue'
      }
    });
  </script>
</body>
</html>
```

3. 数据绑定指令

1. v-text 填充纯文本
   ① 相比插值表达式更加简洁
2. v-html 填充HTML片段
   ① 存在安全问题
   ② 本网站内部数据可以使用，来自第三方的数据不可以用
3. v-pre 填充原始信息
   ① 显示原始信息，跳过编译过程（分析编译过程）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{msg}}</div>
    <div v-text='msg'></div>
    <div v-html='msg1'></div>
    <div v-pre>{{msg}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      1、v-text指令用于将数据填充到标签中，作用于插值表达式类似，但是没有闪动问题
      2、v-html指令用于将HTML片段填充到标签中，但是可能有安全问题
      3、v-pre用于显示原始信息
    */
```

```
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Hello Vue',
        msg1: '<h1>HTML</h1>'
      }
    });
  </script>
</body>
</html>
```

4. 数据响应式

1. 如何理解响应式
   ① html5中的响应式（屏幕尺 寸的变化导致样式的变化）
   ② 数据的响应式（数据的变化导致页面内容的变化）
2. 什么是数据绑定
   ① 数据绑定：将数据填充到标签中
3. v-once 只编译一次
   ① 显示内容之后不再具有响应式功能

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{msg}}</div>
    <div v-once>{{info}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      v-once的应用场景：如果显示的信息后续不需要再修改，你们可以使用v-once，这样可以提高性
能。
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Hello Vue',
        info: 'nihao'
      }
    });
  </script>
</body>
</html>
```

3.3 双向数据绑定指令

1. 什么是双向数据绑定?

2. 双向数据绑定分析

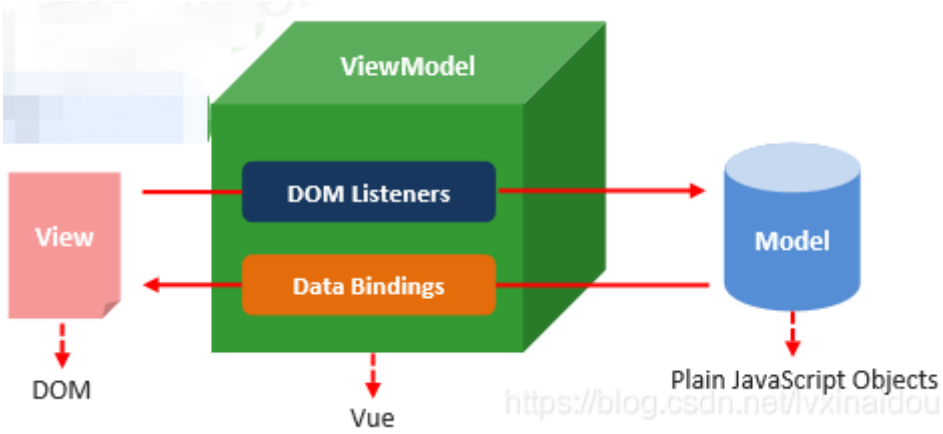
v-model指令用法

3. MVVM设计思想

① M(model)

② V(view)

③ VM(View-Model)




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{msg}}</div>
    <div>
      <input type="text" v-model='msg'>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      双向数据绑定
      1、从页面到数据
      2、从数据到页面
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Hello Vue'
      }
    });
  </script>
</body>
</html>
```

3.4 事件绑定

1. Vue如何处理事件?

v-on指令用法

```
 <input type='button'  v-on:click='num++'/>
```

v-on简写形式

```
<input type='button' @click='num++'/>
```

2. 事件函数的调用方式

直接绑定函数名称

```
<button v-on:click='say'>Hello</button>
```

调用函数

```
<button v-on:click='say()'>Say hi</button>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <div id="app">
        <div>{{num}}</div>
        <div>
            <button v-on:click='num++'>点击</button>
            <button @click='num++'>点击1</button>
            <button @click='handle'>点击2</button>
            <button @click='handle()'>点击3</button>
        </div>
    </div>
    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        var vm = new Vue({
            el: '#app',
            data: {

                num: 0

            }, // 注意点:  这里不要忘记加逗号
            // methods  中 主要是定义一些函数
            methods: {

                handle: function() {
                    // 这里的this是Vue的实例对象+
                    console.log(this === vm)
                        //    在函数中 想要使用data里面的数据 一定要加this
                    this.num++;
                }
```

```
            }

        });
    </script>
</body>

</html>
```

3. 事件函数参数传递

普通参数和事件对象

```
<button v-on:click='say("hi",$event)'>Say hi</button>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <div id="app">
        <div>{{num}}</div>
        <div>
            <!-- 如果事件直接绑定函数名称，那么默认会传递事件对象作为事件函数的第一个参数 -->
            <button v-on:click='handle1'>点击1</button>
            <!-- 2、如果事件绑定函数调用，那么事件对象必须作为最后一个参数显示传递，
                 并且事件对象的名称必须是$event
            -->
            <button v-on:click='handle2(123, 456, $event)'>点击2</button>
        </div>
    </div>
    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        var vm = new Vue({
            el: '#app',
            data: {
                num: 0
            },
            methods: {
                handle1: function(event) {
                    console.log(event.target.innerHTML)
                },
                handle2: function(p, p1, event) {
                    console.log(p, p1)
                    console.log(event.target.innerHTML)
                    this.num++;
                }
            }
        });
    </script>
</body>

</html>
```

4. 事件修饰符

.stop 阻止冒泡

```
<a v-on:click.stop="handle">跳转</a>
```

.prevent 阻止默认行为

```
<a v-on:click.prevent="handle">跳转</a>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{num}}</div>
    <div v-on:click='handle0'>
      <button v-on:click.stop='handle1'>点击1</button>
    </div>
    <div>
      <a href="http://www.baidu.com" v-on:click.prevent='handle2'>百度</a>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      事件绑定-事件修饰符
    */
    var vm = new Vue({
      el: '#app',
      data: {
        num: 0
      },
      methods: {
        handle0: function(){
          this.num++;
        },
        handle1: function(event){
          // 阻止冒泡
          // event.stopPropagation();
        },
        handle2: function(event){
          // 阻止默认行为
          // event.preventDefault();
        }
      }
    });
  </script>
</body>
</html>
```

5. 按键修饰符

.enter 回车键

```
<input v-on:keyup.enter='submit'>
```

.esc 退出键

```
<input v-on:keyup.delete='handle'>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <form action="">
      <div>
        用户名:
        <input type="text" v-on:keyup.delete='clearContent' v-model='uname'>
      </div>
      <div>
        密码:
        <input type="text" v-on:keyup.f1='handleSubmit' v-model='pwd'>
      </div>
      <div>
        <input type="button" v-on:click='handleSubmit' value="提交">
      </div>
    </form>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      事件绑定-按键修饰符
    */
    Vue.config.keyCodes.f1 = 113
    var vm = new Vue({
      el: '#app',
      data: {
        uname: '',
        pwd: '',
        age: 0
      },
      methods: {
        clearContent:function(){
          // 按delete键的时候，清空用户名
          this.uname = '';
        },
        handleSubmit: function(){
          console.log(this.uname,this.pwd)
        }
      }
    });
  </script>
</body>
```

```
</html>
```

6. 自定义按键修饰符

全局 config.keyCodes 对象

```
Vue.config.keyCodes.f1 = 112
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <input type="text" v-on:keyup.aaa='handle' v-model='info'>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      事件绑定-自定义按键修饰符
      规则：自定义按键修饰符名字是自定义的，但是对应的值必须是按键对应event.keyCode值
    */
    Vue.config.keyCodes.aaa = 65
    var vm = new Vue({
      el: '#app',
      data: {
        info: ''
      },
      methods: {
        handle: function(event){
          console.log(event.keyCode)
        }
      }
    });
  </script>
</body>
</html>
```

案例：简单计算器


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
```

```
</head>
<body>
  <div id="app">
    <h1>简单计算器</h1>
    <div>
      <span>数值A:</span>
      <span>
        <input type="text" v-model='a'>
      </span>
    </div>
    <div>
      <span>数值B:</span>
      <span>
        <input type="text" v-model='b'>
      </span>
    </div>
    <div>
      <button v-on:click='handle'>计算</button>
    </div>
    <div>
      <span>计算结果:</span>
      <span v-text='result'></span>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      简单计算器案例
    */
    var vm = new Vue({
      el: '#app',
      data: {
        a: '',
        b: '',
        result: ''
      },
      methods: {
        handle: function(){
          // 实现计算逻辑
          this.result = parseInt(this.a) + parseInt(this.b);
        }
      }
    });
  </script>
</body>
</html>
```

3.5 属性绑定

1. Vue如何动态处理属性?
   v-bind指令用法

```
<a v-bind:href='url'>跳转</a>
```

缩写形式

```
<a :href='url'>跳转</a>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <a v-bind:href="url">百度</a>
    <a :href="url">百度1</a>
    <button v-on:click='handle'>切换</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      属性绑定
    */
    var vm = new Vue({
      el: '#app',
      data: {
        url: 'http://www.baidu.com'
      },
      methods: {
        handle: function(){
          // 修改URL地址
          this.url = 'http://itcast.cn';
        }
      }
    });
  </script>
</body>
</html>
```

2. v-model的低层实现原理分析

```
<input v-bind:value="msg" v-on:input="msg=$event.target.value">
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <div id="app">
        <div>{{msg}}</div>
        <input type="text" v-bind:value="msg" v-on:input='handle'>
        <input type="text" v-bind:value="msg" v-
on:input='msg=$event.target.value'>
        <input type="text" v-model='msg'>
```

```
    </div>
    <script type="text/javascript" src="js/vue.js"></script>
    <script type="text/javascript">
        /*
              v-model指令的本质

            */
        var vm = new Vue({
            el: '#app',
            data: {
                msg: 'hello'
            },
            methods: {
                handle: function(event) {
                    // 使用输入域中的最新的数据覆盖原来的数据
                    this.msg = event.target.value;
                }
            }
        });
    </script>
</body>

</html>
```

3.6 样式绑定

1. class样式处理
   对象语法

```
<div v-bind:class="{ active: isActive }"></div>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">
    .active {
      border: 1px solid red;
      width: 100px;
      height: 100px;
    }
    .error {
      background-color: orange;
    }
  </style>
</head>
<body>
  <div id="app">
    <div v-bind:class="{active: isActive,error: isError}">
      测试样式
    </div>
    <button v-on:click='handle'>切换</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
```

```
    /*
      样式绑定

    */
    var vm = new Vue({
      el: '#app',
      data: {
        isActive: true,
        isError: true
      },
      methods: {
        handle: function(){
          // 控制isActive的值在true和false之间进行切换
          this.isActive = !this.isActive;
          this.isError = !this.isError;
        }
      }
    });
  </script>
</body>
</html>
```

数组语法

```
<div v-bind:class="[activeClass, errorClass]"></div>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">
    .active {
      border: 1px solid red;
      width: 100px;
      height: 100px;
    }
    .error {
      background-color: orange;
    }
  </style>
</head>
<body>
  <div id="app">
    <div v-bind:class='[activeClass, errorClass]'>测试样式</div>
    <button v-on:click='handle'>切换</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      样式绑定

    */
    var vm = new Vue({
```

```
      el: '#app',
      data: {
        activeClass: 'active',
        errorClass: 'error'
      },
      methods: {
        handle: function(){
          this.activeClass = '';
          this.errorClass = '';
        }
      }
    });
  </script>
</body>
</html>
```

class细节用法

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">
    .active {
      border: 1px solid red;
      width: 100px;
      height: 100px;
    }
    .error {
      background-color: orange;
    }
    .test {
      color: blue;
    }
    .base {
      font-size: 28px;
    }
  </style>
</head>
<body>
  <div id="app">
    <div v-bind:class='[activeClass, errorClass, {test: isTest}]'>测试样式</div>
    <div v-bind:class='arrClasses'></div>
    <div v-bind:class='objClasses'></div>
    <div class="base" v-bind:class='objClasses'></div>

    <button v-on:click='handle'>切换</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      样式绑定相关语法细节：
      1、对象绑定和数组绑定可以结合使用
      2、class绑定的值可以简化操作
      3、默认的class如何处理？默认的class会保留
```

```
    */
    var vm = new Vue({
      el: '#app',
      data: {
        activeClass: 'active',
        errorClass: 'error',
        isTest: true,
        arrClasses: ['active','error'],
        objClasses: {
          active: true,
          error: true
        }
      },
      methods: {
        handle: function(){
          // this.isTest = false;
          this.objClasses.error = false;
        }
      }
    });
  </script>
</body>
</html>
```

2. style样式处理
   对象语法

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize }"></div>
```

数组语法

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

</head>
<body>
  <div id="app">
    <div v-bind:style='{border: borderStyle, width: widthStyle, height:
heightStyle}'></div>
    <div v-bind:style='objStyles'></div>
    <div v-bind:style='[objStyles, overrideStyles]'></div>
    <button v-on:click='handle'>切换</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      样式绑定之内联样式Style:
```

```
    */
    var vm = new Vue({
      el: '#app',
      data: {
        borderStyle: '1px solid blue',
        widthStyle: '100px',
        heightStyle: '200px',
        objStyles: {
          border: '1px solid green',
          width: '200px',
          height: '100px'
        },
        overrideStyles: {
          border: '5px solid orange',
          backgroundColor: 'blue'
        }
      },
      methods: {
        handle: function(){
          this.heightStyle = '100px';
          this.objStyles.width = '100px';
        }
      }
    });
  </script>
</body>
</html>
```

3.7 分支循环结构

1. 分支结构

1. v-if
2. v-else
3. v -else-if
4. v-show

2. v-if与v-show的区别

1. v-if控制元素是否渲染到页面
2. v-show控制元素是否显示（已经渲染到了页面）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

</head>
<body>
  <div id="app">
    <div v-if='score>=90'>优秀</div>
    <div v-else-if='score<90&&score>=80'>良好</div>
    <div v-else-if='score<80&&score>60'>一般</div>
    <div v-else>比较差</div>
    <div v-show='flag'>测试v-show</div>
```

```
    <button v-on:click='handle'>点击</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      分支结构

      v-show的原理：控制元素样式是否显示 display:none
    */
    var vm = new Vue({
      el: '#app',
      data: {
        score: 10,
        flag: false
      },
      methods: {
        handle: function(){
          this.flag = !this.flag;
        }
      }
    });
  </script>
</body>
</html>
```

3. 循环结构

v-for遍历数组

```
<li v-for='item in list'>{{item}}</li>
```

```
<li v-for='(item,index) in list'>{{item}} + '---' +{{index}}</li>
```

key的作用：帮助Vue区分不同的元素，从而提高性能

```
<li :key='item.id' v-for='(item,index) in list'>{{item}} + '---' {{index}}</li>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

</head>
<body>
  <div id="app">
    <div>水果列表</div>
    <ul>
      <li v-for='item in fruits'>{{item}}</li>
      <li v-for='(item, index) in fruits'>{{item + '---' + index}}</li>
      <li :key='item.id' v-for='(item, index) in myFruits'>
        <span>{{item.ename}}</span>
        <span>-----</span>
```

```
        <span>{{item.cname}}</span>
      </li>

    </ul>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      循环结构-遍历数组
    */
    var vm = new Vue({
      el: '#app',
      data: {
        fruits: ['apple', 'orange', 'banana'],
        myFruits: [{
          id: 1,
          ename: 'apple',
          cname: '苹果'
        },{
          id: 2,
          ename: 'orange',
          cname: '橘子'
        },{
          id: 3,
          ename: 'banana',
          cname: '香蕉'
        }]
      }
    });
  </script>
</body>
</html>
```

v-for遍历对象

```
<div v-for='(value, key, index) in object'></div>
```

v-if和v-for结合使用

```
<div v-if='value==12' v-for='(value, key, index) in object'></div>
```

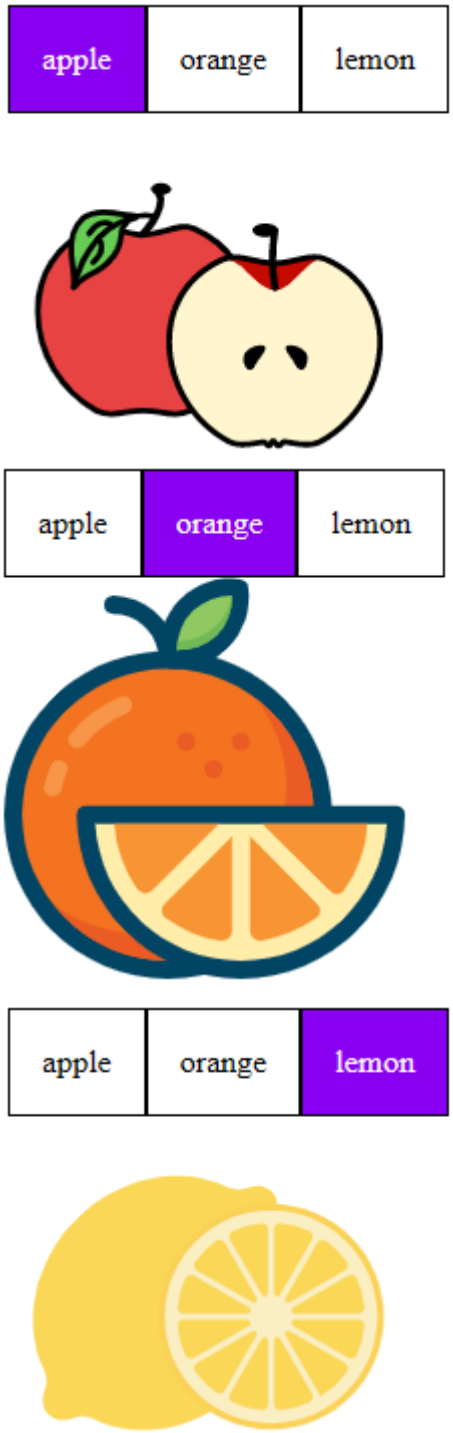
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

</head>
<body>
  <div id="app">
    <div v-if='v==13' v-for='(v,k,i) in obj'>{{v + '---' + k + '---' + i}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
```

```
    // 使用原生js遍历对象
    var obj = {
      uname: 'lisi',
      age: 12,
      gender: 'male'
    }
    for(var key in obj) {
      console.log(key, obj[key])
    }
    /*
      循环结构
    */
    var vm = new Vue({
      el: '#app',
      data: {
        obj: {
          uname: 'zhangsan',
          age: 13,
          gender: 'female'
        }
      }
    });
  </script>
</body>
</html>
```

4. 基础案例

运行效果：




```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Tab Demo</title>
    <script src="../vue.js"></script>
    <style>
        .tab ul {
            list-style: none;
            padding: 0;
            margin: 0;
            font-size: 0px;
```

```
        }

        .tab ul li {
            display: inline-block;
            height: 32px;
            line-height: 32px;
            text-align: center;
            padding: 10px 16px;
            border: 1px solid black;
            font-size: 16px;
            cursor: pointer;
        }

        .tab div.current {
            display: block;
        }

        .tab div {
            display: none;
            width: 300px;
            height: 300px;
        }

        .active {
            background-color: #8800ef;
            color: #fff;
        }
    </style>
</head>

<body>
    <div id="app">
        <div class="tab">
            <ul>

                <li :key="item.id" v-for="(item, index) in list" v-
bind:class="currentIndex === index?'active': ''" @click="change(index)">
{{item.title}}</li>
            </ul>

            <div :key="item.id" v-for="(item, index) in list"
:class="currentIndex === index?'current': ''">
                <img v-bind:src="item.path">
            </div>

        </div>

    </div>

</body>
<script>
    let vue = new Vue({
        el: '#app',
        data: {
            list: [{
                id: 0,
                title: 'apple',
                path: '../image/apple.png'
```

```
            }, {
                id: 1,
                title: 'orange',
                path: '../image/orange.png'
            }, {
                id: 2,
                title: 'lemon',
                path: '../image/lemon.png'
            }],
            currentIndex: 0
        },
        methods: {
            change: function(index) {
                this.currentIndex = index;
            }

        }
    })
</script>

</html>
```

5. Vue常用特性

5.1 常用特性概览

1. 表单操作
2. 自定义指令
3. 计算属性
4. 侦听器
5. 过滤器
6. 生命周期

5.2 表单操作

1. 基于Vue的表单操作
    1. Input 单行文本
    2. textarea 多行文本
    3. select 下拉多选
    4. radio 单选框
    5. checkbox 多选框

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">

  form div {
    height: 40px;
    line-height: 40px;
  }
  form div:nth-child(4) {
```

```
    height: auto;
  }
  form div span:first-child {
    display: inline-block;
    width: 100px;
  }
  </style>
</head>
<body>
  <div id="app">
    <form action="http://itcast.cn">
      <div>
        <span>姓名：</span>
        <span>
          <input type="text" v-model='uname'>
        </span>
      </div>
      <div>
        <span>性别：</span>
        <span>
          <input type="radio" id="male" value="1" v-model='gender'>
          <label for="male">男</label>
          <input type="radio" id="female" value="2" v-model='gender'>
          <label for="female">女</label>
        </span>
      </div>
      <div>
        <span>爱好：</span>
        <input type="checkbox" id="ball" value="1" v-model='hobby'>
        <label for="ball">篮球</label>
        <input type="checkbox" id="sing" value="2" v-model='hobby'>
        <label for="sing">唱歌</label>
        <input type="checkbox" id="code" value="3" v-model='hobby'>
        <label for="code">写代码</label>
      </div>
      <div>
        <span>职业：</span>
        <select v-model='occupation' multiple>
          <option value="0">请选择职业...</option>
          <option value="1">教师</option>
          <option value="2">软件工程师</option>
          <option value="3">律师</option>
        </select>
      </div>
      <div>
        <span>个人简介：</span>
        <textarea v-model='desc'></textarea>
      </div>
      <div>
        <input type="submit" value="提交" @click.prevent='handle'>
      </div>
    </form>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      表单基本操作
    */
```

```
    var vm = new Vue({
      el: '#app',
      data: {
        uname: 'lisi',
        gender: 2,
        hobby: ['2','3'],
        // occupation: 3
        occupation: ['2','3'],
        desc: 'nihao'
      },
      methods: {
        handle: function(){
          // console.log(this.uname)
          // console.log(this.gender)
          // console.log(this.hobby.toString())
          // console.log(this.occupation)
          console.log(this.desc)

        }
      }
    });
  </script>
</body>
</html>
```

2. 表单域修饰符

    1. number：转化为数值
    2. trim：去掉开始和结尾的空格
    3. lazy：将input事件切换为change事件

```
<input v-model.number="age" type="number">
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <input type="text" v-model.number='age'>
    <input type="text" v-model.trim='info'>
    <input type="text" v-model.lazy='msg'>
    <div>{{msg}}</div>
    <button @click='handle'>点击</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      表单域修饰符
    */
    var vm = new Vue({
      el: '#app',
      data: {
        age: '',
```

```
        info: '',
        msg: ''
      },
      methods: {
        handle: function(){
          // console.log(this.age + 13)
          // console.log(this.info.length)
        }
      }
    });
  </script>
</body>
</html>
```

5.3 自定义指令

1. 为何需要自定义指令?
内置指令不满足需求

2. 自定义指令的语法规则（获取元素焦点）

```
Vue.directive('focus' {
        inserted: function(el) { // 获取元素的焦点 el.focus();
    }
})
```

3. 自定义指令用法

```
<input type="text" v-focus>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <input type="text" v-focus>
    <input type="text">
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      自定义指令
    */
    Vue.directive('focus', {
      inserted: function(el){
        // el表示指令所绑定的元素
        el.focus();
      }
    });
    var vm = new Vue({
      el: '#app',
```

```
      data: {

      },
      methods: {
        handle: function(){

        }
      }
    });
  </script>
</body>
</html>
```

带参数的自定义指令（改变元素背景色）

```
Vue.directive('color', {
    inserted: function(el, binding) {
        el.style.backgroundColor = binding.value.color;
    }
})
```

4. 指令的用法

```
<input type="text" v-color='{color:"orange"}'>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <input type="text" v-color='msg'>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      自定义指令-带参数
    */
    Vue.directive('color', {
      bind: function(el, binding){
        // 根据指令的参数设置背景色
        // console.log(binding.value.color)
        el.style.backgroundColor = binding.value.color;
      }
    });
    var vm = new Vue({
      el: '#app',
      data: {
        msg: {
          color: 'blue'
        }
```

```
      },
      methods: {
        handle: function(){

        }
      }
    });
  </script>
</body>
</html>
```

5. 局部指令

```
directives: {
    focus: { // 指令的定义 inserted: function (el) {
            el.focus()
        }
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <input type="text" v-color='msg'>
    <input type="text" v-focus>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      自定义指令-局部指令
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: {
          color: 'red'
        }
      },
      methods: {
        handle: function(){

        }
      },
      directives: {
        color: {
          bind: function(el, binding){
            el.style.backgroundColor = binding.value.color;
          }
        },
        focus: {
```

```
          inserted: function(el) {
            el.focus();
          }
        }
      }
    });
  </script>
</body>
</html>
```

5.4 计算属性

1. 为何需要计算属性?
表达式的计算逻辑可能会比较复杂，使用计算属性可以使模板内容更加简洁
2. 计算属性的用法

```
computed: {
    reversedMessage: function () {
        return this.msg.split('').reverse().join('')
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{msg}}</div>
    <div>{{reverseString}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      计算属性
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Nihao'
      },
      computed: {
        reverseString: function(){
          return this.msg.split('').reverse().join('');
        }
      }
    });
  </script>
</body>
</html>
```

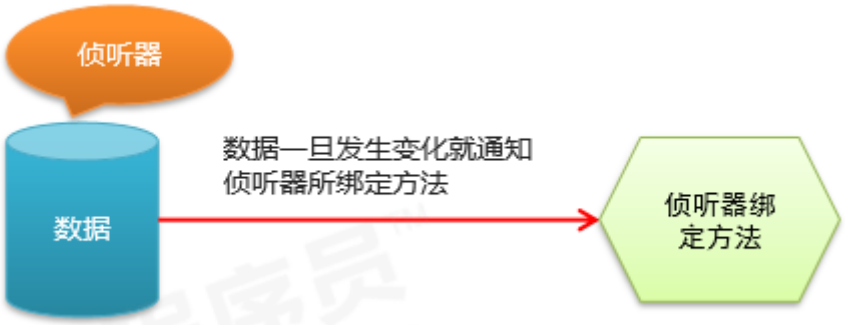
3. 计算属性与方法的区别

1. 计算属性是基于它们的依赖进行缓存的
2. 方法不存在缓存

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{reverseString}}</div>
    <div>{{reverseString}}</div>
    <div>{{reverseMessage()}}</div>
    <div>{{reverseMessage()}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      计算属性与方法的区别:计算属性是基于依赖进行缓存的，而方法不缓存
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Nihao',
        num: 100
      },
      methods: {
        reverseMessage: function(){
          console.log('methods')
          return this.msg.split('').reverse().join('');
        }
      },
      computed: {
        reverseString: function(){
          console.log('computed')
          // return this.msg.split('').reverse().join('');
          var total = 0;
          for(var i=0;i<=this.num;i++){
            total += i;
          }
          return total;
        }
      }
    });
  </script>
</body>
</html>
```

5.5 侦听器

1. 侦听器的应用场景
   数据变化时执行异步或开销较大的操作



2. 侦听器的用法

```
watch: {
    firstName: function(val){ // val表示变化之后的值 this.fullName = val +
this.lastName;
    },
    lastName: function(val) {
    this.fullName = this.firstName + val;
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>
      <span>名：</span>
      <span>
        <input type="text" v-model='firstName'>
      </span>
    </div>
    <div>
      <span>姓：</span>
      <span>
        <input type="text" v-model='lastName'>
      </span>
    </div>
    <div>{{fullName}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      侦听器
    */
    var vm = new Vue({
      el: '#app',
      data: {
        firstName: 'Jim',
        lastName: 'Green',
        // fullName: 'Jim Green'
      },
      computed: {
```

```
        fullName: function(){
          return this.firstName + ' ' + this.lastName;
        }
      },
      watch: {
        // firstName: function(val) {
        //   this.fullName = val + ' ' + this.lastName;
        // },
        // lastName: function(val) {
        //   this.fullName = this.firstName + ' ' + val;
        // }
      }
    });
  </script>
</body>
</html>
```

3.侦听器案例

需求：输入框中输入姓名，失去焦点时验证是否存在，如果已经存在，提示从新输入，如果不存在，提示可以使用。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>
      <span>用户名：</span>
      <span>
        <input type="text" v-model.lazy='uname'>
      </span>
      <span>{{tip}}</span>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      侦听器
      1、采用侦听器监听用户名的变化
      2、调用后台接口进行验证
      3、根据验证的结果调整提示信息
    */
    var vm = new Vue({
      el: '#app',
      data: {
        uname: '',
        tip: ''
      },
      methods: {
        checkName: function(uname) {
```

```
            // 调用接口，但是可以使用定时任务的方式模拟接口调用
            var that = this;
            setTimeout(function(){
              // 模拟接口调用
              if(uname == 'admin') {
                that.tip = '用户名已经存在，请更换一个';
              }else{
                that.tip = '用户名可以使用';
              }
            }, 2000);
          }
        },
        watch: {
          uname: function(val){
            // 调用后台接口验证用户名的合法性
            this.checkName(val);
            // 修改提示信息
            this.tip = '正在验证...';
          }
        }
      });

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

5.6 过滤器

1. 过滤器的作用是什么?
格式化数据，比如将字符串格式化为首字母大写，将日期格式化为指定的格式等
2. 自定义过滤器

```
Vue.filter('过滤器名称', function(value){ // 过滤器业务逻辑 })
```

3. 过滤器的使用

```
<div>{{msg | upper}}</div>
<div>{{msg | upper | lower}}</div>
<div v-bind:id="id | formatId"></div>
```

4. 局部过滤器

```
filters:{
    capitalize: function(){}
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
```

```
  <div id="app">
    <input type="text" v-model='msg'>
    <div>{{msg | upper}}</div>
    <div>{{msg | upper | lower}}</div>
    <div :abc='msg | upper'>测试数据</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      过滤器
      1、可以用与插值表达式和属性绑定
      2、支持级联操作
    */
    // Vue.filter('upper', function(val) {
    //   return val.charAt(0).toUpperCase() + val.slice(1);
    // });
    Vue.filter('lower', function(val) {
      return val.charAt(0).toLowerCase() + val.slice(1);
    });
    var vm = new Vue({
      el: '#app',
      data: {
        msg: ''
      },
      filters: {
        upper: function(val) {
          return val.charAt(0).toUpperCase() + val.slice(1);
        }
      }
    });
  </script>
</body>
</html>
```

5. 带参数的过滤器

```
Vue.filter('format', function(value, arg1){ // value就是过滤器传递过来的参数 })
```

6. 过滤器的使用

```
<div>{{date | format('yyyy-MM-dd')}}</div>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{date | format('yyyy-MM-dd hh:mm:ss')}}</div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
```

```
    /*
      过滤器案例：格式化日期

    */
    // Vue.filter('format', function(value, arg) {
    //   if(arg == 'yyyy-MM-dd') {
    //     var ret = '';
    //     ret += value.getFullYear() + '-' + (value.getMonth() + 1) + '-' +
value.getDate();
    //     return ret;
    //   }
    //   return value;
    // })
    Vue.filter('format', function(value, arg) {
      function dateFormat(date, format) {
          if (typeof date === "string") {
              var mts = date.match(/(\/Date\((\d+)\)\/)/);
              if (mts && mts.length >= 3) {
                  date = parseInt(mts[2]);
              }
          }
          date = new Date(date);
          if (!date || date.toUTCString() == "Invalid Date") {
              return "";
          }
          var map = {
              "M": date.getMonth() + 1, //月份
              "d": date.getDate(), //日
              "h": date.getHours(), //小时
              "m": date.getMinutes(), //分
              "s": date.getSeconds(), //秒
              "q": Math.floor((date.getMonth() + 3) / 3), //季度
              "S": date.getMilliseconds() //毫秒
          };

          format = format.replace(/([yMdhmsqS])+/g, function(all, t) {
              var v = map[t];
              if (v !== undefined) {
                  if (all.length > 1) {
                      v = '0' + v;
                      v = v.substr(v.length - 2);
                  }
                  return v;
              } else if (t === 'y') {
                  return (date.getFullYear() + '').substr(4 - all.length);
              }
              return all;
          });
          return format;
      }
      return dateFormat(value, arg);
    })
    var vm = new Vue({
      el: '#app',
      data: {
        date: new Date()
      }
    });
```

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

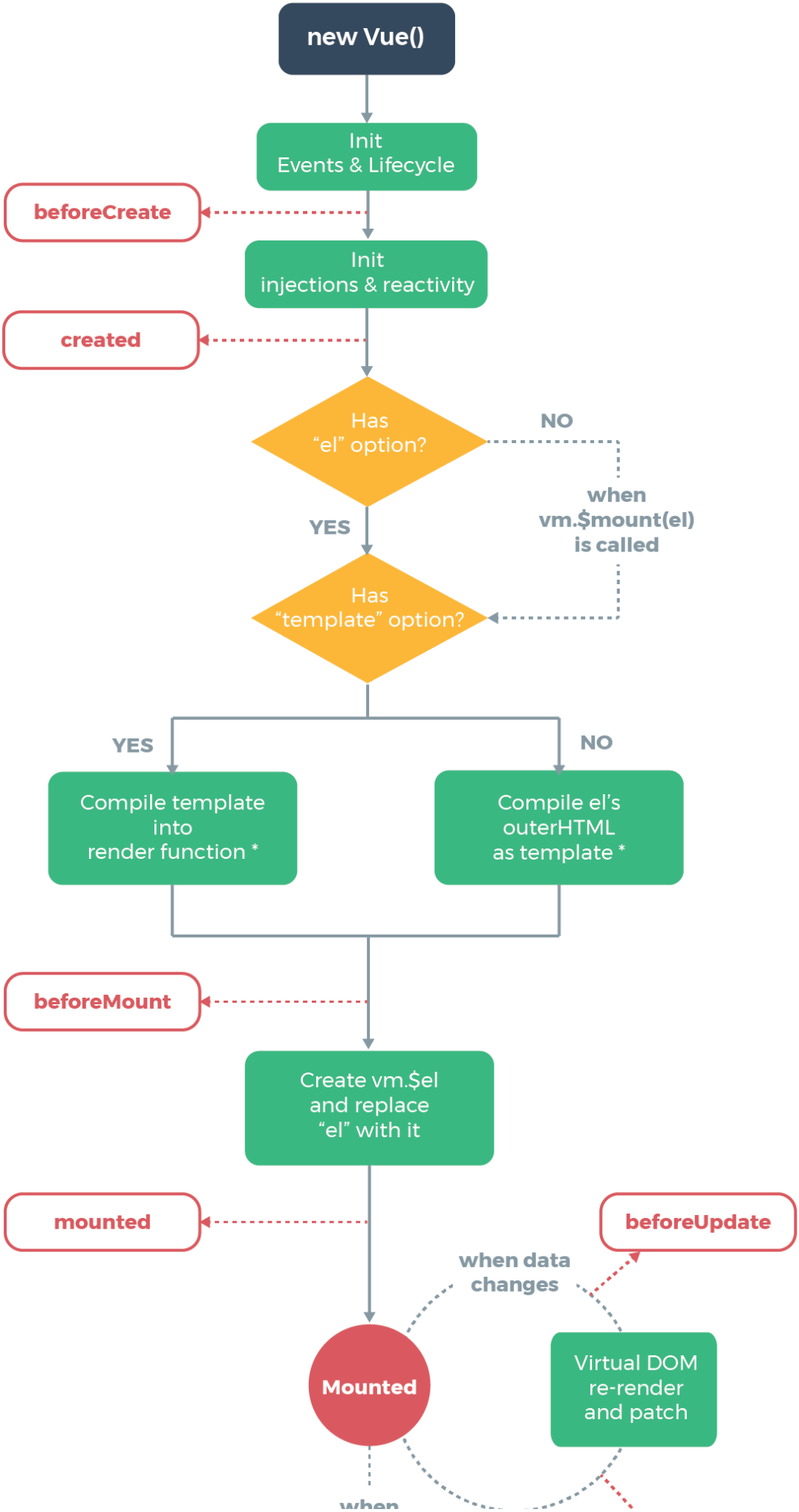
5.7 生命周期

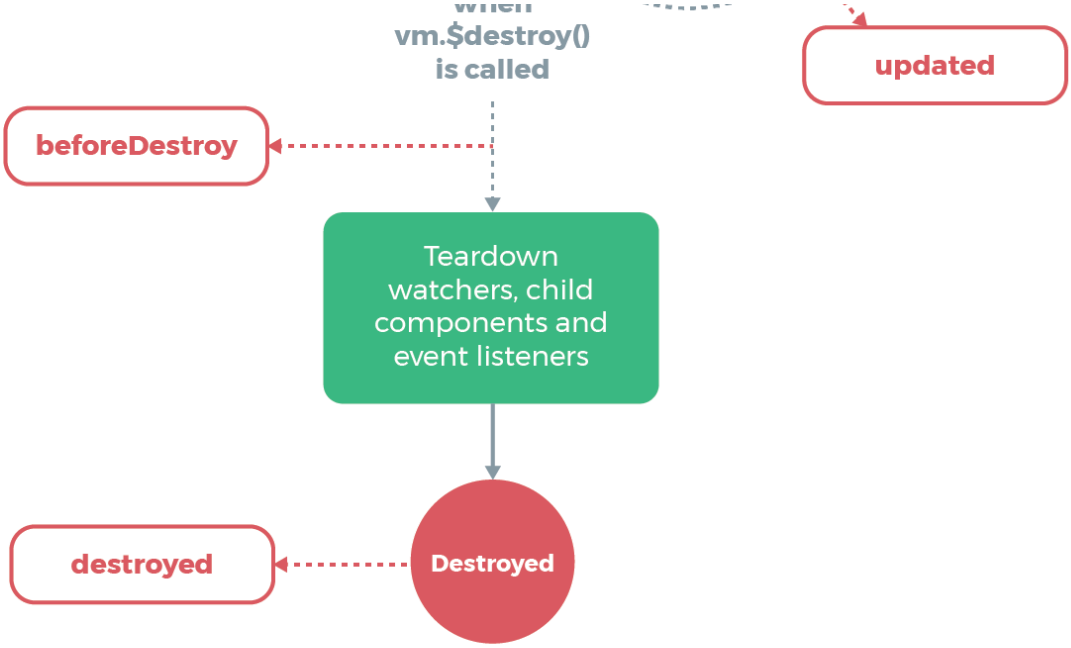
1. 主要阶段

1. 挂载（初始化相关属性） ① beforeCreate ② created ③ beforeMount ④ mounted
2. 更新（元素或组件的变更操作） ① beforeUpdate ② updated
3. 销毁（销毁相关属性） ① beforeDestroy ② destroyed

2. Vue实例的产生过程

① beforeCreate 在实例初始化之后，数据观测和事件配置之前被调用。
② created 在实例创建完成后被立即调用。
③ beforeMount在挂载开始之前被调用。
④ mounted el被新创建的vm.$el替换，并挂载到实例上去之后调用该钩子。
⑤ beforeUpdate 数据更新时调用，发生在虚拟DOM打补丁之前。
⑥ updated 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。
⑦ beforeDestroy 实例销毁之前调用。
⑧ destroyed 实例销毁后调用。


new Vue()
Init
Events & Lifecycle
beforeCreate
Init
injections & reactivity
created
Has
“el” option?
NO
when
vm.$mount(el)
is called
YES
Has
“template” option?
YES
NO
Compile template
into
render function *
Compile el’s
outerHTML
as template *
beforeMount
Create vm.$el
and replace
“el” with it
mounted
beforeUpdate
when data
changes
Mounted
Virtual DOM
re-render
and patch
when



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


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <div>{{msg}}</div>
    <button @click='update'>更新</button>
    <button @click='destroy'>销毁</button>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      Vue实例的生命周期

    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: '生命周期'
      },
      methods: {
        update: function(){
          this.msg = 'hello';
        },
        destroy: function(){
          this.$destroy();
        }
```

```
      },
      beforeCreate: function(){
        console.log('beforeCreate');
      },
      created: function(){
        console.log('created');
      },
      beforeMount: function(){
        console.log('beforeMount');
      },
      mounted: function(){
        console.log('mounted');
      },
      beforeUpdate: function(){
        console.log('beforeUpdate');
      },
      updated: function(){
        console.log('updated');
      },
      beforeDestroy: function(){
        console.log('beforeDestroy');
      },
      destroyed: function(){
        console.log('destroyed');
      }
    });
  </script>
</body>
</html>
```

6. 综合案例

案例：补充知识（数组相关API）

1. 变异方法(修改原有数据)

1. push()
2. pop()
3. shift()
4. un shift()
5. spl ice()
6. sort()
7. reverse()

2. 替换数组(生成新的数组)

1. filter() 2. concat() 3. slice()

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
```

```
  <div id="app">
    <div>
      <span>
        <input type="text" v-model='fname'>
        <button @click='add'>添加</button>
        <button @click='del'>删除</button>
        <button @click='change'>替换</button>
      </span>
    </div>
    <ul>
      <li :key='index' v-for='(item,index) in list'>{{item}}</li>
    </ul>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      Vue数组操作
      1、变异方法：会影响数组的原始数据的变化。
      2、替换数组：不会影响原始的数组数据，而是形成一个新的数组。
    */
    var vm = new Vue({
      el: '#app',
      data: {
        fname: '',
        list: ['apple','orange','banana']
      },
      methods: {
        add: function(){
          this.list.push(this.fname);
        },
        del: function(){
          this.list.pop();
        },
        change: function(){
          this.list = this.list.slice(0,2);
        }
      }
    });
  </script>
</body>
</html>
```

3. 修改响应式数据

1. Vue.set(vm.items, indexOfItem, newValue)
2. vm.$set(vm.items, indexOfItem, newValue)
   ① 参数一表示要处理的数组名称
   ② 参数二表示要处理的数组的索引
   ③ 参数三表示要处理的数组的值

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
```

```
</head>
<body>
  <div id="app">
    <ul>
      <li v-for='item in list'>{{item}}</li>
    </ul>
    <div>
      <div>{{info.name}}</div>
      <div>{{info.age}}</div>
      <div>{{info.gender}}</div>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      动态处理响应式数据

    */
    var vm = new Vue({
      el: '#app',
      data: {
        list: ['apple', 'orange', 'banana'],
        info: {
          name: 'lisi',
          age: 12
        }
      },
    });
    // vm.list[1] = 'lemon';
    // Vue.set(vm.list, 2, 'lemon');
    vm.$set(vm.list, 1, 'lemon');

    // vm.info.gender = 'male';
    vm.$set(vm.info, 'gender', 'female');


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

案例：图书管理

运行效果


```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>图书管理</title>
    <script src="../vue.js"></script>
    <style>
        .grid {
            margin: auto;
            width: 500px;
            text-align: center;
        }

        .grid table {
            width: 100%;
            border-collapse: collapse;
        }

        .grid th,
        td {
            padding: 10;
            border: 1px dashed orange;
            height: 35px;
            line-height: 35px;
        }

        .grid th,
        .book div {
            background-color: orange;
            border-top: 1px solid;
        }

        .book div {
            padding: 10px 0px;
```

```
        }

        .grid .total {
            height: 30px;
            line-height: 30px;
            background-color: orange;
            border-top: 1px solid
        }
    </style>
</head>

<body>
    <div id="app">
        <div class="grid">
            <div class="book">
                <h1>图书管理</h1>
                <div>
                    <label for="id">编号:</label>
                    <input type="text" id="id" v-model="id"
:disabled="isIdDisabled" v-focus>
                    <label for="name">名称:</label>
                    <input type="text" id="name" v-model="name">
                    <button @click="addHandle()" :disabled="isAllowAdd">添加
</button>
                </div>
            </div>
            <div class="total">
                <span>图书总数:</span>
                <span>{{total}}</span>
            </div>
            <table>
                <thead>
                    <th>编号</th>
                    <th>名称</th>
                    <th>时间</th>
                    <th>操作</th>
                </thead>
                <tbody>
                    <tr :key="item.id" v-for="(item, index) in books">
                        <td>{{item.id}}</td>
                        <td>{{item.name}}</td>
                        <td>{{item.date | format('yyyy-MM-dd hh:mm:ss')}}</td>
                        <td>
                            <a href="#" @click.prevent="editHandle(index)"> 编辑
</a>
                            <span>|</span>
                            <a href="#" @click.prevent="deleteHandle(index)"> 删
除</a>
                        </td>
                    </tr>
                    <!-- <tr>
                        <td>2</td>
                        <td>水浒</td>
                        <td></td>
                        <td>
                            <a>添加 | </a>
                            <a> 删除 </a>
                        </td>
```

```
                    </tr>
                    <tr>
                        <td>3</td>
                        <td>红楼</td>
                        <td></td>
                        <td>
                            <a>添加 | </a>
                            <a> 删除 </a>
                        </td>
                    </tr>
                    <tr>
                        <td>4</td>
                        <td>西游</td>
                        <td></td>
                        <td>
                            <a>添加 | </a>
                            <a> 删除 </a>
                        </td>
                    </tr> -->
                </tbody>
            </table>
        </div>
    </div>

</body>
<script>
    let vue = new Vue({
        el: '#app',
        data: {
            id: '',
            name: '',
            isIdDisabled: false,
            isAllowAdd: false,
            books: [],
            // books: [{
            //     id: 1,
            //     name: '三国',
            //     date: 2525609975000
            // }, {
            //     id: 2,
            //     name: '水浒',
            //     date: 2525609975000
            // }, {
            //     id: 3,
            //     name: '红楼',
            //     date: 2525609975000
            // }, {
            //     id: 4,
            //     name: '西游',
            //     date: 2525609975000
            // }]

        },
        filters: {
            format: function(date, format) {
                console.log(date);

                function dateFormat(date, format) {
```

```
                    if (typeof date === "string") {
                        var mts = date.match(/(\/Date\((\d+)\)\/)/);
                        if (mts && mts.length >= 3) {
                            date = parseInt(mts[2]);
                        }
                    }
                    date = new Date(date);
                    if (!date || date.toUTCString() == "Invalid Date") {
                        return "";
                    }
                    var map = {
                        "M": date.getMonth() + 1, //月份
                        "d": date.getDate(), //日
                        "h": date.getHours(), //小时
                        "m": date.getMinutes(), //分
                        "s": date.getSeconds(), //秒
                        "q": Math.floor((date.getMonth() + 3) / 3), //季度
                        "S": date.getMilliseconds() //毫秒
                    };

                    format = format.replace(/([yMdhmsqS])+/g, function(all, t) {
                        var v = map[t];
                        if (v !== undefined) {
                            if (all.length > 1) {
                                v = '0' + v;
                                v = v.substr(v.length - 2);
                            }
                            return v;
                        } else if (t === 'y') {
                            return (date.getFullYear() + '').substr(4 -
all.length);
                        }
                        return all;
                    });
                    return format;
                }
                return dateFormat(date, format);
            }
        },
        directives: {
            focus: {
                inserted: function(el) {
                    el.focus();
                }
            }
        },
        computed: {
            total: function() {
                return this.books.length;
            }
        },
        watch: {
            name: function(value) {
                let flag = this.books.some(function(item) {
                    return item.name === value;
                });
                this.isAllowAdd = flag ? true : false;
            }
```

```
        },
        mounted: function() {
            //初始化后通过接口调用数据
            let data = [{
                id: 1,
                name: '三国',
                date: 2525609975000
            }, {
                id: 2,
                name: '水浒',
                date: 2525609975000
            }, {
                id: 3,
                name: '红楼',
                date: 2525609975000
            }, {
                id: 4,
                name: '西游',
                date: 2525609975000
            }];
            this.books = data;
            console.log(this.books);
        },
        methods: {
            addHandle: function() {
                if (this.isIdDisabled) {
                    //edit
                    this.books[this.id].name = this.name;
                    this.isIdDisabled = false;
                    //清空表单
                    this.id = '';
                    this.name = '';
                } else {
                    //add
                    //添加
                    let bookItem = {};
                    bookItem.id = this.id;
                    bookItem.name = this.name;
                    bookItem.date = new Date();
                    this.books.push(bookItem);
                    //清空表单
                    this.id = '';
                    this.name = '';
                }


            },
            editHandle: function(index) {
                this.id = this.books[index].id;
                this.name = this.books[index].name;
                this.isIdDisabled = true;
            },
            deleteHandle: function(index) {
                this.books.splice(index, 1);
            }

        }
    });
```

```
</script>

</html>
```

github地址

核心代码
学习地址