任务一 Vue.js

1.Vue.js

1.1 Vue.js 介绍

1.1.1 Vue.js是什么?

Vue (读音 /vju:/，类似于 view) 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计 为可以自底向上逐层应用.

Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一 方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

自底向上逐层应用：作为渐进式框架要实现的目标就是方便项目增量开发(即插即用)。

> 官方网站: https://cn.vuejs.org/v2/guide/ 作者 尤雨溪是中国人.

1.1.2 为甚么使用Vue?

1. 声明式渲染: 前后端分离是未来趋势
2. 渐进式框架: 适用于各种业务需求
3. 简单易学: 国人开发,中文文档,不存在语言障碍,易于理解和学习

1.2 Vue.js 基础

1.2.1 Vue.js的使用

1. 在html页面使用script引入vue.js的库即可使用。

```
远程CDN
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
本地
<script src="vue.min.js"></script>
```

```
2. Vue-CLI脚手架:使用vue.js官方提供的CLI脚本架很方便去创建vue.js工程雏形
```

1.2.2 入门程序

创建一个vuetest目录, 并且在目录下创建 01_vue入门程序.html 文件.

代码编写步骤:

1、定义html，引入vue.js

2、定义app div，此区域作为vue的接管区域

3、定义Vue实例，接管app区域。

4、定义model（数据对象）

5、在app中展示数据

```
<!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>

    <!-- 1.创建HTML文件，  引入vue.js 有两种方式-->

    <!-- 第一种 引入 vue.js的CDN地址 -->
    <!-- <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js">
</script> -->

    <!--  第二种 本地导入 -->
    <script src="vue.min.js"></script>


</head>
<body>
    <!-- 2. 定义app div，此区域作为vue的接管区域 -->
    <div id="app">
        <!-- {{}} 双括号是VUE中的差值表达式,将表达式的值输出到HTML页面  -->
        {{name}}
    </div>
</body>

<script>
    //3. 创建vue实例
    var VM = new Vue({
        //定义 Vue实例挂载的元素节点,表示vue接管该div
        el:'#app',
        //4.定义model模型数据对象
        data:{
            name:"哈拉少"
        }
    });
</script>

</html>
```

1. {{}}: 插值表达式

1. 插值表达式的作用?

   通常用来获取Vue实例中定义的数据(data)

   属性节点中 不能够使用插值表达式

2. el: 挂载点

1. el的作用？

   定义 Vue实例挂载的元素节点,表示vue接管该区域

2. Vue的作用范围是什么？

   Vue会管理el选项命中的元素,及其内部元素

3. el选择挂载点时,是否可以使用其他选择器？

   可以,但是建议使用 ID选择器

4. 是否可以设置其他的DOM元素进行关联？

   可以但是建议选择DIV, 不能使用HTML和Body标签

3. data: 数据对象

1. Vue中用到的数据定义在data中
2. data中可以写复杂类型
3. 渲染复杂类型数据的时候,遵守js语法

```
<body>
    <!-- 此区域作为vue的接管区域 -->
    <div id="app">
        {{name}} <br>
        {{school.name}} {{school.mobile}}<br>
        <ul>
            <li>{{names[0]}}</li>
            <li>{{names[1]}}</li>
            <li>{{names[2]}}</li>
        </ul>
    </div>
</body>

<script>

    //创建vue实例
    var VM = new Vue({
        el:'#app',
        data:{
            name:"雷霆八嘎",
            //对象类型数据
            school:{
                name:"拉钩教育",
                mobile:"1001001"
            },
            //数组类型
            names:["小斌","张百万","刘能"]
        }
    });

</script>
```

1.2.3 声明式渲染的好处

Vue中的声明式渲染,简单理解就是我们声明数据,Vue帮我们将数据渲染到HTML.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div id="app">
      <h2>{{name}}</h2>
    </div>
```

```
  </body>


  <!--
  jQuery中，如果 DOM 发生变化，js代码也需要做相应的改变，高耦合 .
  <script src="./js/jquery-1.8.3.min.js"></script>
  <script>
    $(document).ready(function () {
      $("#app").append("<h2>Hello Word! !</h2>");
    });
  </script> -->

  <!-- 在用 Vue中，只需要定义好展示数据，并把它放在 DOM 合适的位置就可以. -->
  <script src="js/vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app", //挂载点
      data: {
        name: "Hello Word! !",
      },
    });
  </script>
</html>
```

1.2.4 Vue常用指令

根据官网的介绍，指令 是带有 v- 前缀的特殊属性。通过指令来操作DOM元素

1. v-text 指令

作用: 获取data数据, 设置标签的内容.

注意: 默认写法会替换全部内容,使用插值表达式{{}}可以替换指定内容.

代码示例

```
<body>
    <div id="app">
        <!-- v-text 获取data数据,设置标签内容,会覆盖之前的内容体-->
        <h2 v-text="message">百度</h2>

        <!-- 使用插值表达式,不会覆盖 -->
        <h2>{{message}}百度</h2>

        <!-- 拼接字符串 -->
        <h2 v-text="message+1"></h2>
        <h2 v-text="message+'abc'"></h2>
    </div>
    <script>
        var VM = new Vue({
            el:"#app",
            data:{
                message:"Java程序员"
            }
```

```
        })

    </script>
</body>

</html>
```

2. v-html 指令

作用: 设置元素的 innerHTML (可以向元素中写入新的标签)

代码示例

```
<body>
    <div id="app">
      <!-- 获取普通文本 -->
      {{message}}
      <h2 v-text="message"></h2>
      <h2 v-html="message"></h2>

      <!-- 设置元素的innerHTML -->
      <h2 v-html="url"></h2>
      <h2 v-text="url"></h2>
    </div>
  </body>
  <script src="./js/vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app",
      data: {
        message: "Java程序员",
        url: "<a href='https://www.baidu.com'>百度一下</a>",
      },
    });
  </script>
```

3. v-on 指令

作用: 为元素绑定事件, 比如: v-on:click,可以简写为 @click="方法"

绑定的方法定义在 VUE实例的, method属性中

语法格式

```
<div id="app">
    <!-- 使用v-on 绑定click 点击事件 -->
    <input type="button" value="点击按钮" v-on:click="方法名">
    <!-- 使用 @符号也可以绑定-->
    <input type="button" value="点击按钮" @click="方法名">
</div>

var VM = new Vue({
    el:"#app",
    //通过methods ,专门存放Vue中的方法
    methods:{
        方法名:function(){
            alert("123!")
        }
```

```
    }
})
```

代码示例

```
<body>
    <div id="app">
        <!-- 使用v-on 绑定click 点击事件 -->
        <input type="button" value="点击按钮" v-on:click="show">

        <!-- 简写 @方式 -->
        <input type="button" value="点击按钮" @click="show">

        <!-- 双击事件 -->
        <input type="button" value="双击击按钮" @dblclick="show">

        <!-- 绑定点击事件 -->
        <h2 @click="changeFood">{{food}}</h2>
    </div>
</body>

<script src="vue.min.js"></script>
<script>
    var VM = new Vue({
        el:"#app",
        data:{
            food:"麻辣小龙虾"
        },
        //通过methods ,专门存放Vue中的方法
        methods:{
            show:function(){
                alert("程序员!")
            },
            changeFood:function(){
                //使用this获取
                console.log(this.food);
                //在VUE中不需要考虑如何更改DOM元素， 重点放在更改数据,数据更新之后,使用数据
的那个元素会同步更新
                this.food+="真好吃!";
            }
        }
    })
</script>
```

4. 计数器案例

1) 编码步骤

1. data中定义数据: 比如 num 值为1
2. methods中添加两个方法: 比如add(递增) ,sub(递减)
3. 使用{{}} 将num设置给 span标签
4. 使用v-on 将add,sub 分别绑定给 + ,- 按钮
5. 累加到10 停止
6. 递减到0 停止

2) 页面准备

```
<body>
    <div id="app">
        <!-- 计算功能区域 -->
        <div>
            <input type="button" class="btn btn_plus">
            <span>{{num}}</span>
            <input type="button" class="btn btn_minus">
        </div>
    </div>
</body>
<script src="vue.min.js"></script>
<script>
    //创建VUE实例
    var VM = new Vue({
        el:"#app",
        data:{
            num:1
        }
    })
</script>
```

3) 案例演示

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="./css/inputNum.css">
</head>
<body>
    <div id="app">
        <!-- 计算功能区域 -->
        <div>
            <input type="button" class="btn btn_plus" @click="add">
            <span>{{num}}</span>
            <input type="button" class="btn btn_minus" @click="sub">
        </div>
    </div>
</body>
<script src="vue.min.js"></script>
<script>
    //创建VUE实例
    var VM = new Vue({
        el:"#app",
        data:{
            num:1
        },
        methods: {
            add:function(){
                //console.log("add");
                if(this.num < 10){
                    this.num++;
                }else{
                    alert("别点啦!最大了!")
                }
```

```
            },
            sub:function(){
                //console.log("sub");
                if(this.num > 0){
                    this.num--;
                }else{
                    alert("别点啦!最小了!")
                }
            }
        }
    })
</script>
</html>
```

4) 案例总结

- 创建VUE实例时: **el(挂载点)， data(数据)，methods(方法)**
- **v-on** 指令的作用是绑定事件,简写为 **@**
- 方法中使用**this**关键字,获取data中的数据
- **v-text** 与 **{{}}** 的作用都是用来 设置元素的文本值

5. v-show指令

作用: v-show指令, 根据真假值,切换元素的显示状态

页面准备

```
<body>
    <div id="app">
        <img src="./img/car.gif" alt="">
    </div>
</body>
<script src="vue.min.js"></script>
<script>
    var VM = new Vue({
        el:"#app"
    })
</script>
```

代码示例

```
<body>
    <div id="app">
      <input type="button" value="切换状态" @click="changeShow" />
      <img v-show="isShow" src="./img/car.gif" />
      <img v-show="age > 18" src="./img/car.gif" />
    </div>
  </body>

  <script src="js/vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app",
      data: {
        isShow: true,
        age: 19,
      },
```

```
      methods: {
        changeShow: function () {
          //触发方法, 对isShow进行取反
          this.isShow = !this.isShow;
        },
      },
    });
  </script>
```

v-show 指令总结

- 原理是修改元素的display,实现显示或者隐藏
- 指令后面的内容,最终会解析为 布尔值
- 值为true 显示, 为false 则隐藏
- 数据改变之后,显示的状态会同步更新

6. v-if 指令

作用: 根据表达值的真假,切换元素的显示和隐藏(操纵dom 元素)

代码示例

```
<body>
    <div id="app">
        <input type="button" value="切换显示状态" @click="changeShow">
        <img v-if="isShow" src="./img/car.gif" alt="">
    </div>
</body>
<script src="./vue.min.js"></script>
<script>
    var VM = new Vue({
        el:"#app",
        data:{
            isShow:false
        },
        methods: {
            changeShow:function(){
                this.isShow = !this.isShow;
            }
        }
    })

</script>
```

v-if 指令总结

- v-if 指令的作用: 根据表达式的真假切换元素的显示状态
- 本质是通过操作dom元素,来切换显示状态
- 表达式为true 元素存在与dom树,为false从dom树中移除
- 频繁切换使用 v-show ,反之使用v-if

7. v-bind 指令

作用: 设置元素的属性 (比如:src,title,class)

```
语法格式:  v-bind:属性名=表达式

<img v-bind:src="imgSrc">

var VM = new Vue({
    el:"#app",
    data:{
        imgSrc:"图片地址"
    }
})

v-bind 可以省略，简写为冒号 :
<img :src="imgSrc">
```

代码示例

```
<body>
    <div id="app">
        <!-- 使用v-bind设置src属性值 -->
        <img v-bind:src="imgSrc" alt="">

        <!-- 简写 设置title -->
        <img :src="imgSrc" alt="" :title="imgTitle">

        <!-- 设置class -->
        <div :style="{ fontSize: size + 'px'}">v-bind指令</div>

    </div>
</body>
<script src="./vue.min.js"></script>
<script>
    var VM = new Vue({
        el:"#app",
        data:{
            imgSrc:"./img/lagou.jpg",
            imgTitle:"拉钩教育",
            size:100
        }
    })
</script>
```

v-bind指令总结

- v-bind 指令的作用是： 为元素绑定属性
- 完整写法 **v-bind:属性名**，可以简写为 **:属性名**

8. v-for 指令

作用: 根据数据生成列表结构

语法结构

```
<div id="app">
    <ul>
        <li v-for="item in arr"></li>
```

```
    </ul>
</div>

var VM = new Vue({
    el:"#app",
    data:{
        arr:[1,2,3,4,5],
        objArr:[
            {name:"tom"},
            {name:"jack"}
        ]
    }
})
```

代码示例

```
<body>
    <div id="app">

        <input type="button" value="添加数据" @click="add">
        <input type="button" value="移除数据" @click="remove">
        <ul>
            <!-- 在li标签中获取数组元素 -->
            <li v-for="(item,index) in arr">
                {{index+1 }}城市: {{item}}
            </li>
        </ul>

        <!-- 使用h2标签显示
            v-for 结合 v-bind一起使用
        -->
        <h2 v-for="p in  persons" v-bind:title="p.name">
            {{p.name}}
        </h2>
    </div>
</body>

<script src="./vue.min.js"></script>
<script>
    var VM = new Vue({
        el:"#app",
        data:{
            //普通数组
            arr:["上海","北京","天津","杭州"],
            //对象数组
            persons:[
                {name:"尼古拉斯·赵四"},
                {name:"莱安纳多·小沈阳"}
            ]
        },
        methods: {
            add:function(){
                //push 添加
                this.persons.push({name:"多利安·刘能"})
            },
            remove:function(){
                this.persons.shift();
```

```
            }
        }
    })
</script>
```

v-for指令总结

- v-for 指令的作用: 根据数据生成列表结构
- 数组经常和 v-for结合使用,数组有两个常用方法:
  - push() 向数组末尾添加一个或多个元素
  - shift() 把数组中的第一个元素删除
- 语法是: (item,index) in 数据
- item和index 可以结合其他指令一起使用
- 数组的长度变化,会同步更新到页面上,是响应式的

9. v-on 指令补充

1. 传递自定义参数 : 函数调用传参
2. 事件修饰符: 对事件触发的方式进行限制

代码示例

```
<body>
    <div id="app">
      <!-- 函数传参 -->
      <input
        type="button"
        value="礼物刷起来"
        @click="showTime(666,'爱你老铁!')"
      />

      <!-- 事件修饰符 指定哪些方式可以触发事件 -->
      <input type="text" @keyup.enter="hi" />
    </div>
  </body>
  <script src="./js/vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app",
      data: {},
      methods: {
        showTime: function (p1, p2) {
          console.log(p1);
          console.log(p2);
        },
        hi: function () {
          alert("你好吗?");
        },
      },
    });
  </script>
```

总结

- 事件绑定方法,可以传入自定义参数
- 定义方法时,需要定义形参,来接收实际的参数

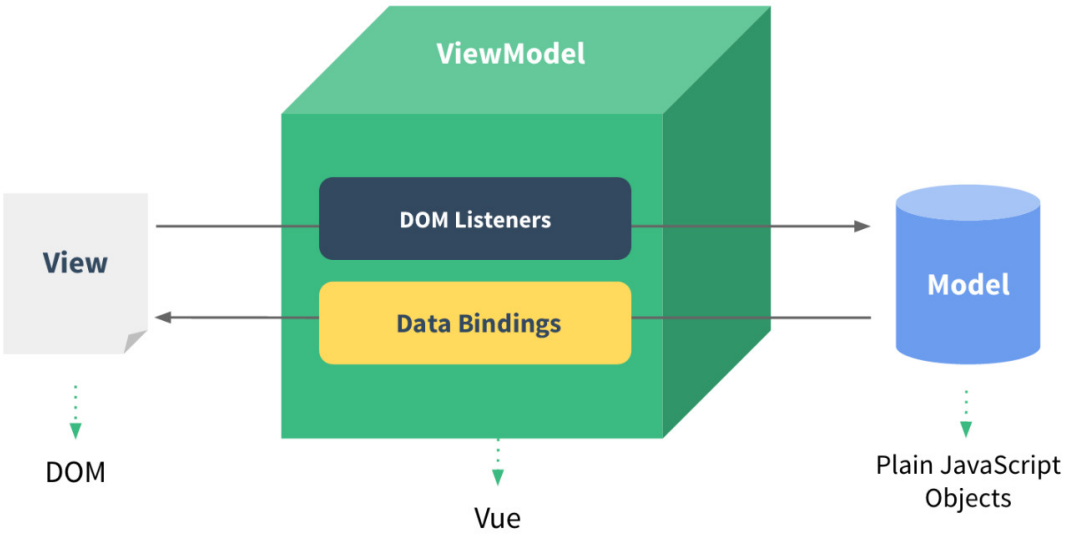
- 事件的后面跟上 .修饰符 可以对事件进行限制
- .enter 可以限制触发的按键为回车
- 事件修饰符有许多 使用时可以查询文档

10. MVVM模式

- MVVM 是Model-View-ViewModel 的缩写，它是一种基于前端开发的架构模式.
- MVVM模式将页面,分层了 M 、V、和VM ,解释为:
  - Model: 负责数据存储
  - View: 负责页面展示
  - View Model: 负责业务逻辑处理（比如Ajax请求等），对数据进行加工后交给视图展示

```
<body>
    <div id="app">
      <!-- View 视图部分 -->
      <h2>{{name}}</h2>
    </div>
  </body>
  <script src="./js/vue.min.js"></script>
  <script>

    //创建的vue实例,就是 VM ViewModel
    var VM = new Vue({
      el: "#app",
      //data就是MVVM模式中的 model
      data: {
        name: "hello",
      },
    });
  </script>
```



- 首先，我们将上图中的DOM Listeners和Data Bindings看作两个工具，它们是实现双向绑定的关键。
  - 从View侧看，ViewModel中的DOM Listeners工具会帮我们监测页面上DOM元素的变化，如果有变化，则更改Model中的数据；
  - 从Model侧看，当我们更新Model中的数据时，Data Bindings工具会帮我们更新页面中的DOM元素。

- MVVM的思想,主要是为了让我们的开发更加的方便,因为MVVM提供了**数据的双向绑定**

11. v-mode 指令

作用: 获取和设置表单元素的值(实现双向数据绑定)

- **双向数据绑定**
  - 单向绑定: 就是把Model绑定到View，当我们用JavaScript代码更新Model时，View就会自动更新。
  - 双向绑定: 用户更新了View，Model的数据也自动被更新了，这种情况就是双向绑定。
- 什么情况下用户可以更新View呢?
  - 填写表单就是一个最直接的例子。当用户填写表单时，View的状态就被更新了，如果此时MVVM框架可以自动更新Model的状态，那就相当于我们把Model和View做了双向绑定:

代码示例

```
<body>
    <div id="app">
      <input type="button" value="修改message" @click="update" />

      <!-- View 视图 -->
      <!-- <input type="text" v-bind:value="message" /> -->

      <!-- v-model 实现双向数据绑定 -->
      <input type="text" v-model="message" />
      <input type="text" v-model="password" />
      <h2>{{message}}</h2>
    </div>
  </body>
  <script src="./js/vue.min.js"></script>
  <script>
    //VM 业务逻辑控制
    var VM = new Vue({
      el: "#app",
      //Model 数据存储
      data: {
        message: "拉钩教育训练营",
        password: 123,
      },
      methods: {
        update: function () {
          this.message = "拉钩";
        },
      },
    });
  </script>
```

v-model指令总结

- v-model 指令的作用是便捷的设置和获取表单元素的值
- 绑定的数据会和表单元素值相关联
- 双向数据绑定

1.2.5 实现简单记事本

1.功能介绍

2.新增内容

步骤

1. 生成列表结构(v-for 数组)
2. 获取用户输入(v-model 双向绑定)
3. 回车,新增数据(v-on .enter事件修饰符)
4. 页面布局不熟悉,可以通过审查元素的方式快速找到元素

```
    <body>
        <!-- VUE示例接管区域 -->
        <section id="app">

            <!-- 输入框 -->
            <header class="header">
            <h1>VUE记事本</h1>

            <!-- v-on 绑定事件 -->
            <input v-model="inputValue" @keyup.enter="add"
                autofocus="autofocus" autocomplete="off" placeholder="输入日程"
class="new-todo"/>
            </header>

            <!-- 列表区域 -->
            <section class="main">
            <ul class="listview">
                <li class="todo" v-for="(item,index) in list">
                <div class="view">
                    <span class="index">{{index+1}}</span> <label>{{item}}
</label>
                    <button class="destroy"></button>
                </div>
                </li>
            </ul>
            </section>
        </section>
    </body>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        var VM = new Vue({
            el:"#app",
            data:{
                list:["写代码","吃饭","睡觉"],
                inputValue:"996还是997"
            },
            methods: {
                //新增方法
                add:function(){
                    //将用户输入的内容添加到list
                    this.list.push(this.inputValue);
                }
            }
        })
    </script>
```

3.删除内容

步骤

1. 点击删除指定的内容(根据索引删除元素)
2. 在methods中添加一个删除的方法,使用splice函数进行删除

```
<!-- 列表区域 -->
<section class="main">
    <ul class="listview">
        <li class="todo" v-for="(item,index) in list">
            <div class="view">
                <span class="index">{{index+1}}</span> <label>{{item}}</label>

                <!-- 删除按钮 -->
                <button class="destroy" @click="remove(index)"></button>
            </div>
        </li>
    </ul>
</section>
```

```
//删除方法
remove:function(index){
    console.log(index);
    //使用splice(元素索引,删除几个) 根据索引删除
    this.list.splice(index,1);
}
```

4.统计操作

步骤

1. 统计页面信息的个数,就是列表中的元素的个数.
2. 获取 list数组的长度,就是信息的个数

```
<!-- 统计和清空 -->
<footer class="footer">
    <span class="todo-count"> <strong>{{list.length}}</strong> items left
</span>
    <button class="clear-completed">
        Clear
    </button>
</footer>
```

总结:

1. 基于数据的开发方式

    2. v-text设置的是文本,可以使用简化方式 {{}}

5.清空数据

步骤:

1. 点击清除所有信息
2. 本质就是清空数组

```
<button class="clear-completed" @click="clear()">Clear</button>
```

```
//清空数组元素
clear:function(){
    this.list=[];
}
```

1.3 axios

1.3.1 Ajax回顾

1.3.1.1 什么是Ajax?

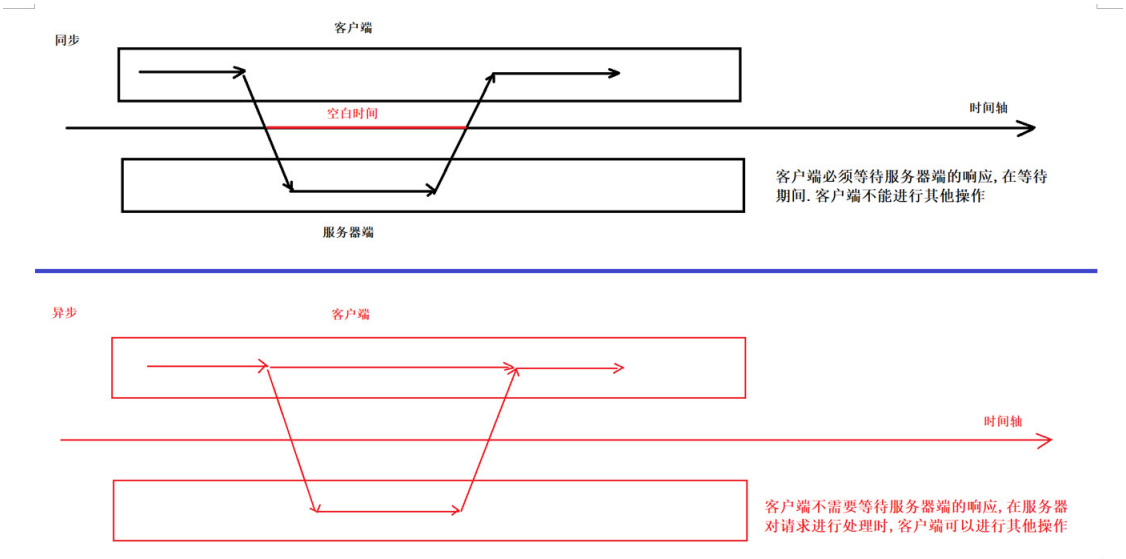
Ajax 是指一种创建交互式网页应用的开发技术。Ajax = 异步 JavaScript 和 XML。

1.3.1.2 Ajax的作用

- Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新（局部更新）。传统的网页如果需要更新内容，必须重载整个网页页面。
- 简单记: Ajax 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术, 维护用户体验性, 进行网页的局部刷新.

1.3.1.3 异步与同步

- 浏览器访问服务器的方式
    - 同步访问: 客户端必须等待服务器端的响应,在等待过程中不能进行其他操作
    - 异步访问: 客户端不需要等待服务的响应,在等待期间,浏览器可以进行其他操作



1.3.1.4 案例演示

ajax.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
</head>
<body>
```

```
    <input type="text" />
    <input type="button" value="Jquery发送异步请求" onclick="run1()" />
</body>
<script src="./jquery-1.8.3.min.js"></script>

<script>
    function run1() {
        //JQuery Ajax方式 发送异步请求
        $.ajax({
            url: "/ajax",
            async:true,
            data: { name: "天青" },
            type: "post",
            dataType:"text",
            success: function (res) {
                console.log(res)
                alert("响应成功" + res);
            },
            error: function () {
                alert("响应失败!");
            }
        });
    }
</script>
</html>
```

servlet

```
@WebServlet("/ajax")
public class AjaxServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

        //1.获取请求数据
        String username = req.getParameter("name");

        //模拟业务操作,造成的延时效果
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        //2.打印username
        System.out.println(username);
        resp.getWriter().write("hello hello");

    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doGet(req, resp);
    }
```

```
}
```

1.3.1 axios介绍

VUE中结合网络数据进行应用的开发

- 目前十分流行网络请求库,专门用来发送请求,其内部还是ajax,进行封装之后使用更加方便
- axios作用: 在浏览器中可以帮助我们完成 ajax异步请求的发送.

> Vue2.0之后，尤雨溪推荐大家用axios替换JQuery ajax

1.3.2 axios入门

使用步骤:

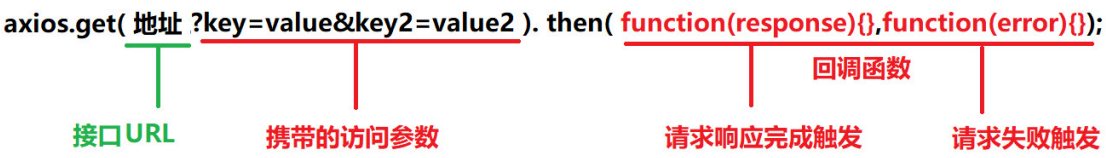
1. 导包

```
<!-- 官网提供的 axios 在线地址 -->
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

2. 请求方式,以GET和POST举例

GET

```
axios.get(地址?key=value&key2=value2).then(function(response){},function(error)
{});
```



要访问的接口的URL, 和key的名称都是接口文档提供的, value值 就是我们要传递的参数值

POST

```
axios.post(地址,{key:value,key2:value2}).then(function(response)
{},function(error){})
```

3. 根据接口文档, 访问测试接口,进行测试

接口1:随机笑话

```
请求地址:https://autumnfish.cn/api/joke/list
请求方法:get
请求参数:num(笑话条数,数字)
响应内容:随机笑话
```

接口2:用户注册

```
请求地址:https://autumnfish.cn/api/user/reg
请求方法:post
请求参数:username(用户名,字符串)
响应内容:注册成功或失败
```

代码示例

```
<body>
    <input type="button" value="get请求" id="get">
    <input type="button" value="post请求" id="post">
</body>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
    /*
        请求地址:https://autumnfish.cn/api/joke/list
        请求方法:get
        请求参数:num(笑话条数,数字)
        响应内容:随机笑话
    */
    document.getElementById("get").onclick=function(){
        axios.get("https://autumnfish.cn/api/joke/list?num=1")
        .then(function(response){
            //请求成功,调用
            console.log(response);

        },function(error){
            //请求失败,调用
            console.log(error)
        });
    }

    /*
        请求地址:https://autumnfish.cn/api/user/reg
        请求方法:post
        请求参数:username(用户名,字符串)
        响应内容:注册成功或失败
    */
    document.getElementById("post").onclick=function(){
        axios.post("https://autumnfish.cn/api/user/reg",{username:"张百万"})
        .then(function(response){
            console.log(response);

        },function(error){
            console.log(error);
        });
    }
</script>
```

1.3.3 axios总结

1. **axios 必须导包才能使用**
2. **使用get或者post方法,就可以发送请求**
3. **then方法中的回调函数,会在请求成功或者请求失败的时候触发**
4. **通过回调函数的形参可以获取响应的内容,或者错误信息**

1.3.4 获取笑话案例

通过vue+axios 完成一个获取笑话的案例.

接口: 随机获取一条笑话

```
请求地址:https://autumnfish.cn/api/joke
请求方法:get
请求参数:无
响应内容:随机笑话
```

代码示例

```
<body>
    <div id="app">
        <input type="button" value="点击获取一个笑话" @click="getJoke">
        <p>{{joke}}</p>
    </div>
</body>
<!-- 引入vue + axios -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>

    /*
        请求地址:https://autumnfish.cn/api/joke
        请求方法:get
        请求参数:无
        响应内容:随机笑话
    */
    var VM = new Vue({
        el:"#app",
        data:{
            joke:"笑笑更健康"
        },
        methods: {
            getJoke:function(){
                console.log(this.joke);//笑笑更健康
                var that = this; //把this保存起来
                //异步访问
                axios.get("https://autumnfish.cn/api/joke").then(
                    function(response){
                        //获取data中的笑话
                        console.log(response.data);
                        //console.log(this.joke); //undefined 没有获取到this
                        that.joke=response.data;
                    },
                    function(error){}
                )
            }
        }

    })

</script>
```

案例总结

1. axios回调函数中this指向已经改变,无法访问data中的数据

2. 解决方案: 将this进行保存,回调函数中直接使用保存的this即可

1.3.5 天气查询案例

1.3.5.1 需求分析

- 功能分析: 回车查询
  - 1.输入内容,点击回车 (v-on.enter)
  - 2.访问接口,查询数据 (axios v-model)
  - 3.返回数据,渲染数据

1.3.5.2 接口文档

```
请求地址:http://wthrcdn.etouch.cn/weather_mini
请求方法:get
请求参数:city (要查询的城市名称)
响应内容:天气信息
```

1.3.5.3 案例演示

自定义JS文件

作为一个标准的应用程序,我们将创建VUE实例的代码,抽取到main.js 文件中

main.js

```
/*
    请求地址:http://wthrcdn.etouch.cn/weather_mini
    请求方法:get
    请求参数:city (要查询的城市名称)
    响应内容:天气信息
*/
var VM = new Vue({
    el:"#app",
    data:{
        city:'',
        //定义空数组接收天气信息
        weatherList:[]
    },
    //编写查询天气方法
    methods: {
        searchWeather:function(){
            console.log("天气查询");
            console.log(this.city);

            //保存this,方便在回调函数中使用
            var that = this;

            //调用接口
            axios.get("http://wthrcdn.etouch.cn/weather_mini?city="+this.city)
            .then(function(respose){
                //console.log(respose);
                //只获取天气数组
                console.log(respose.data.data.forecast);
                that.weatherList = respose.data.data.forecast;
            },function(error){});
        }
```

```
    }

})
```

```
<body>
    <div class="wrap" id="app">
      <div class="search_form">
        <div class="logo">天气查询</div>
        <div class="form_group">
          <!-- 3.绑定点击事件,回车触发,通过v-model绑定数据 -->
          <input v-model="city" @keyup.enter="searchWeather" type="text"
class="input_txt" placeholder="请输入要查询的城市"/>
          <button class="input_sub">回车查询</button>
        </div>

      </div>
      <ul class="weather_list">
        <!-- 遍历天气信息 -->
        <li v-for="item in weatherList">
          <div class="info_type"><span class="iconfont">{{item.type}}</span>
</div>
          <div class="info_temp">
            <b>{{item.low}}</b>
            ~
            <b>{{item.high}}</b>
          </div>
          <div class="info_date"><span>{{item.date}}</span></div>
        </li>
      </ul>
    </div>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <!-- 官网提供的 axios 在线地址 -->
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <!-- 2.引入自己的js -->
    <script src="./js/main.js"></script>
  </body>
```

1.3.5.4 案例总结

1. 应用的逻辑代码,建议与页面进行分离,使用单独的JS编写
2. axios回调函数中的 this的指向改变,无法正常使用, 需要另外保存一份
3. 服务器返回的数据比较的复杂时,获取数据时要注意层级结构

1.3.6 解决页面闪烁问题

我们发现访问天气预报案例页面时, 使用插值表达式的地方出现了闪烁问题,如何解决呢?

v-cloak指令

作用: 解决插值表达式闪烁问题

当网络较慢，网页还在加载 Vue.js ，而导致 Vue 来不及渲染，这时页面就会显示出 Vue 源代码。我们可以使用 v-cloak 指令来解决这一问题。

1) 添加样式

```
<style>
    /* 通过属性选择器,设置 添加了v-cloak */
    [v-cloak] {
        display: none;
    }
</style>
```

2) 在id为app的div中添加 v-cloak

```
<div class="wrap" id="app" v-cloak>
```

1.4 computed 计算属性

1.4.1 什么是计算属性

在Vue应用中，在模板中双向绑定一些数据或者表达式，但是表达式如果过长，或者逻辑更为复杂时，就会变得臃肿甚至难以维护和阅读，比如下面的代码:

```
<div>
    写在双括号中的表达式太长了,不利于阅读
    {{text.split(',').reverse().join(',')}}
</div>.

将这段操作text.split(',').reverse().join(',')  放到计算属性中,最终返回一个结果值就可以
```

computed 的作用: 减少运算次数, 缓存运算结果. 运用于重复相同的计算.

1.4.1 代码示例

```
<body>
    <div id="app">
      <!-- <h1>{{a*b}}</h1>
      <h1>{{a*b}}</h1>  -->

      <!-- <h1>{{res()}}</h1>
      <h1>{{res()}}</h1> -->

      <h1>{{res2}}</h1>
      <h1>{{res2}}</h1>
    </div>
  </body>
  <script src="vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app",
      data: {
        a: 10,
        b: 20,
      },
      methods: {
        res: function () {
          console.log("res方法执行");
          return this.a * this.b;
        },
      },
```

```
      //使用计算属性进行优化 减少运算次数,用于重复相同的运算
      computed: {
        res2: function () {
          console.log("res2方法执行");
          return this.a * this.b;
        },
      },
    });
  </script>
```

1.4.2 computed总结

1. 定义函数也可以实现与 计算属性相同的效果,都可以简化运算。
2. 不同的是**计算属性是基于它们的响应式依赖进行缓存的。**只在相关响应式依赖发生改变时它们才会重新求值。

1.5 filter 过滤器

1.5.1 什么是过滤器

过滤器是对即将显示的数据做进一步的筛选处理，然后进行显示，值得注意的是过滤器并没有改变原来的数据，只是在原数据的基础上产生新的数据。

数据加工车间,对值进行筛选加工.

1.5.2 过滤器使用位置

1. 双括号插值内

```
{{ msg | filterA }}  msg是需要处理的数据, filterA是过滤器, | 这个竖线是管道,通过这个管道将数据传输给过滤器进行过滤 加工操作
```

2. v-bind绑定的值的地方。

```
<h1 v-bind:id=" msg | filterA"> {{ msg }} </h1>
```

1.5.3 过滤器

1.局部过滤器

需求: 通过过滤器给电脑价格前面 添加一个符号¥

```
<body>
    <div id="app">
      <p>电脑价格: {{price | addIcon}}</p>
    </div>
  </body>
  <script src="./vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app", //挂载点
      data: {
        price: 200,
      },
      methods: {}, //方法
      computed: {}, //计算属性
```

```
    //局部过滤器
    filters: {
      //处理函数,value = price ,是固定参数
      addIcon(value) {
        return "¥" + value;
      },
    },
  });
</script>
```

2.全局过滤器

需求: 将用户名开头字母大写

```
<body>
  <div id="app">
    <p>{{user.name | changeName}}</p>
  </div>
</body>
<script src="./vue.min.js"></script>
<script>
  //在创建Vue实例之前 创建全局过滤器
  Vue.filter("changeName", function (value) {
    //将姓名开头字母大写,然后再重新拼接
    return value.charAt(0).toUpperCase() + value.slice(1);
  });

  var VM = new Vue({
    el: "#app", //挂载点
    data: {
      user: { name: "tom" },
    },
  });
</script>
```

1.5.4 总结

1. 过滤器常用来处理文本格式化的操作。过滤器可以用在两个地方：**双花括号插值和 v-bind 表达式**
2. 过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示

1.6 watch 侦听器

1.6.1 什么是侦听器

Vue.js 提供了一个方法 watch，它用于观察Vue实例上的数据变动。

作用: 当你有一些数据需要随着其它数据变动而变动时，可以使用侦听属性

1.6.2 案例演示

需求: 监听姓名变化,实时显示


```
<body>
    <div id="app">
      <label>名：<input type="text" v-model="firstName" /></label>
      <label>姓：<input type="text" v-model="lastName" /></label>
      {{fullNameComputed}}
    </div>
  </body>
  <script>
    var app = new Vue({
      el: "#app",
      data: {
        firstName: "",
        lastName: "",
        fullName: "",
      },
      //监听，程序在运行的时候，实时监听事件
      watch: {
        //参数说明：1、新值，2、旧值
        firstName(newValue, oldValue) {
          this.fullName = newValue + " " + this.lastName;
        },
        lastName(newValue, oldValue) {
          this.fullName = this.firstName + " " + newValue;
        },
      },
      computed: {
        fullNameComputed() {
          return this.firstName + " " + this.lastName;
        },
      },
    });
  </script>
```

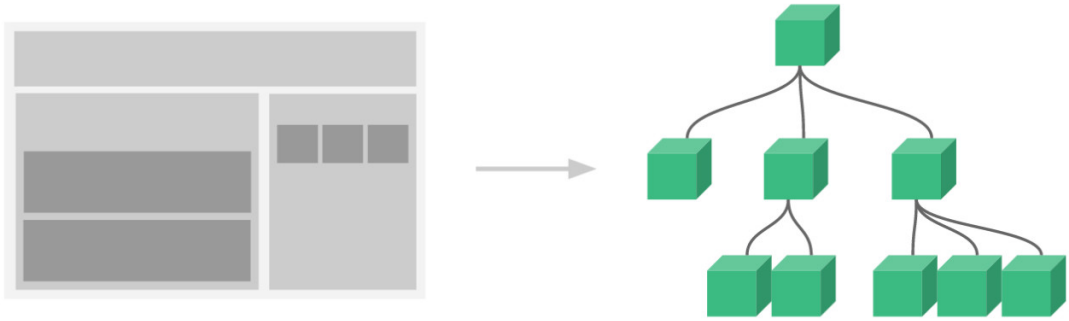
1.7 Component 组件

1.7.1 组件介绍

组件（Component）是自定义封装的功能。在前端开发过程中，经常出现多个网页的功能是重复的，而且很多不同的页面之间，也存在同样的功能。

我们将相同的功能进行抽取,封装为组件,这样，前端人员就可以在组件化开发时，只需要书写一次代码，随处引入即可使用。

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



vue的组件有两种: 全局组件 和 局部组件

1.7.2 全局组件

语法格式:

```
Vue.component("组件名称", {
    template: "html代码",  // 组件的HTML结构代码
    data(){ //组件数据
        return {}
    },
    methods: {  // 组件的相关的js方法
        方法名(){
            // 逻辑代码
        }
    }
})
```

注意:

1. 组件名以小写开头，采用短横线分割命名: 例如 **hello-Word**
2. 组件中的data 必须是一个函数,注意与Vue实例中的data区分
3. 在template模板中, 只能有一个根元素

```
  <body>
    <div id="app">
      <!-- 使用组件,可以使用多次 -->
      <lagou-header></lagou-header>
      <lagou-header></lagou-header>
      <lagou-header></lagou-header>
    </div>
  </body>
  <script src="./vue.min.js"></script>
  <script>
    //全局组件
    Vue.component("lagou-header", {
      //组件的命名一般使用短横线方式,组件中的模板只能有一个根元素
      template: "<div>头部组件的HTML代码<h1 @click='hello'>{{msg}}</h1><div>",
      data() {
        //组件中的data是一个函数
        return {
          msg: "hello这里是组件中的data数据",
        };
```

```
    },
    methods: {
      hello() {
        alert("嗨 你好!");
      },
    },
  });

  var VM = new Vue({
    el: "#app",
    data: {},
    methods: {},
  });
</script>
```

1.7.3 局部组件

相比起全局组件，局部组件只能在同一个实例内才能被调用。局部组件的写法和全局组件差不多。 **唯一不同就是：局部组件要写在Vue实例里面。**

```
new Vue({
    el: "#app",
    components: {
        组件名: {
            // 组件结构
            template: "HTML代码",
            // data数据
            data() { return { msg:"xxxx" };},
        },
    },
});
```

注意:

创建局部组件，注意 components，注意末尾有 ‘s’，而全局组件是不用+ ‘s’ 的。这意味着，components 里可以创建多个组件。

```
<body>
  <div id="app">
    <web-msg></web-msg>
  </div>
</body>
<script src="./vue.min.js"></script>
<script>
  var VM = new Vue({
    el: "#app",
    components: {
      //组件名称
      "web-msg": {
        //组件内容
        template: "<div><h1>{{msg1}}</h1><h1>{{msg2}}</h1></div>",
        data() {
          return {
            msg1: "开发ing...",
            msg2: "开发完成!",
          };
```

```
        },
      },
    },
  });
</script>
```

1.7.4 组件与模板分离

由于把html语言写在组件里面很不方便，也不太好看所以将它们分开写。

```
<body>
  <div id="app">
    <web-msg></web-msg>
  </div>

  <!-- 将模板写在HTML中，给模板指定一个ID -->
  <template id="tmp1">
    <div>
      <button @click="show">{{msg}}</button>
    </div>
  </template>
</body>

<script src="./vue.min.js"></script>
<script>
  var VM = new Vue({
    el: "#app",
    components: {
      "web-msg": {
        template: "#tmp1",
        data() {
          return {
            msg: "点击查询",
          };
        },
        methods: {
          show() {
            alert("正在查询,请稍等...");
          },
        },
      },
      "web-msg2": {},
    },
  });
</script>
```

总结:

1. 上面这种写法，浏览器会把 html 里的 template 标签过滤掉。所以 template 标签的内容是不会在页面中展示的。直到它被 JS 中的 Vue 调用。
2. 在 html 中，template 标签一定要有一个 id，因为通过 id 是最直接被选中的。 data 和 methods 等 参数，全部都要放到 Vue 实例里面写

1.8 Vue生命周期

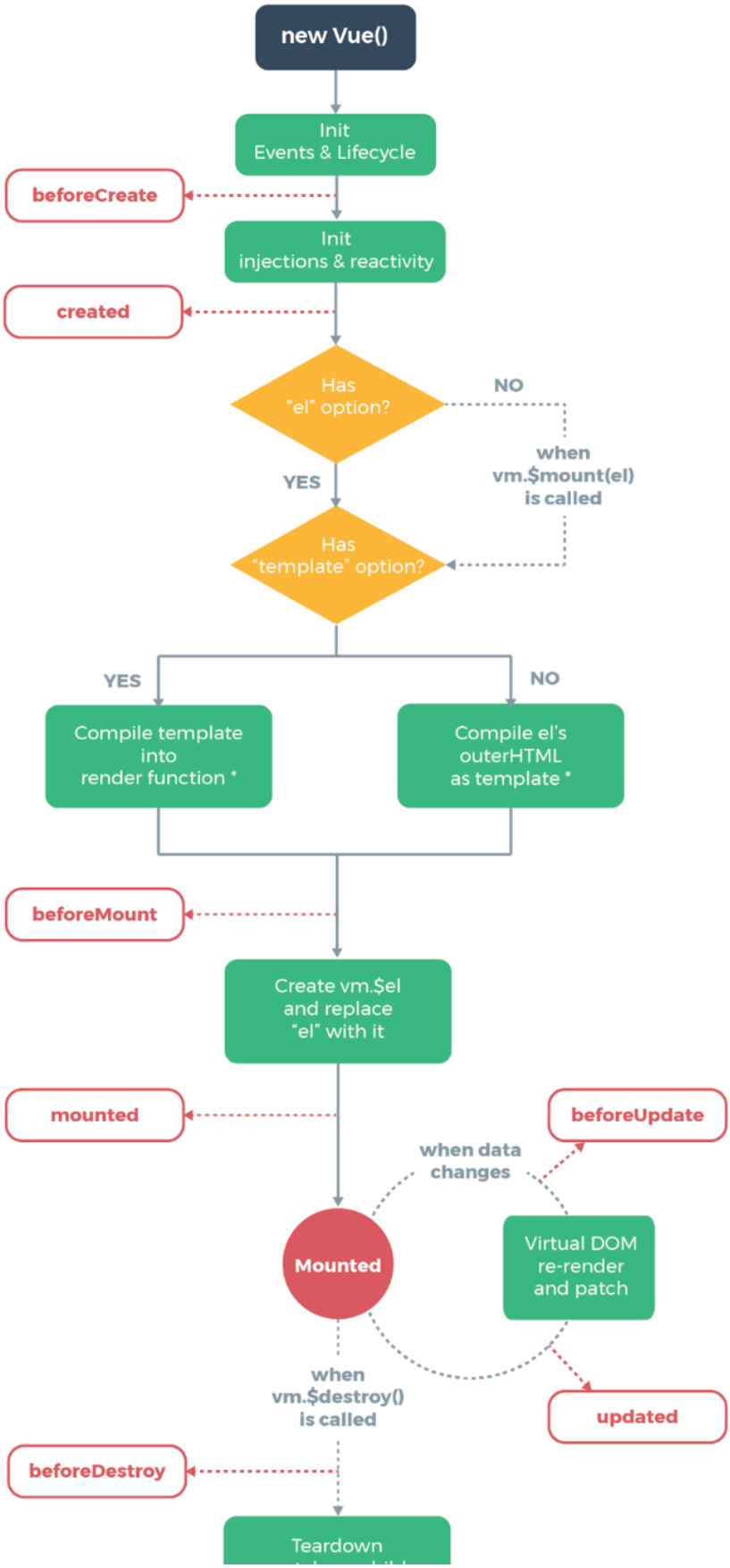
1.8.1 生命周期图示

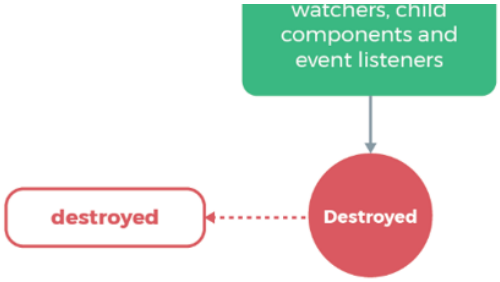
每个Vue实例在被**创建**之前都要经过一系列的初始化过程,这个过程就是vue的生命周期

了解生命周期的好处:

1. 找错误
2. 解决需求

下图展示了实例的生命周期。你不需要立马弄明白所有的东西，不过随着你的不断学习和使用，它的参考价值会越来越高。





1.8.2 钩子函数介绍

生命周期中的钩子函数

钩子函数：钩子函数是在一个事件触发的时候，在系统级捕获到了他，然后做一些操作

函数	说明
beforeCreate()	在创建Vue实例之前,可以执行这个方法. 例如 加载动画操作
created()	实例创建完成,属性绑定好了,但是DOM还没有生成.
beforeMount()	模板已经在内存中编辑完成了，尚未被渲染到页面中.
mounted()	内存中的模板已经渲染到页面，用户已经可以看见内容.
beforeUpdate()	数据更新的前一刻 , 组件在发生更新之前,调用的函数
updated()	updated执行时，内存中的数据已更新，并且页面已经被渲染
beforeDestroy ()	钩子函数在实例销毁之前调用
destroyed ()	钩子函数在Vue 实例销毁后调用

1.8.3 案例演示

```
<body>
    <div id="app">
      <button @click="next">获取下一句</button>
      <h2 id="msg">{{message}}</h2>
    </div>
  </body>
  <script src="./vue.min.js"></script>
  <script>
    var VM = new Vue({
      el: "#app",
      data: {
        message: "想当年,金戈铁马",
      },
      methods: {
        next() {
          this.message = "气吞万里如虎!";
        },
        show() {
          alert("show方法执行!");
        },
      },
```

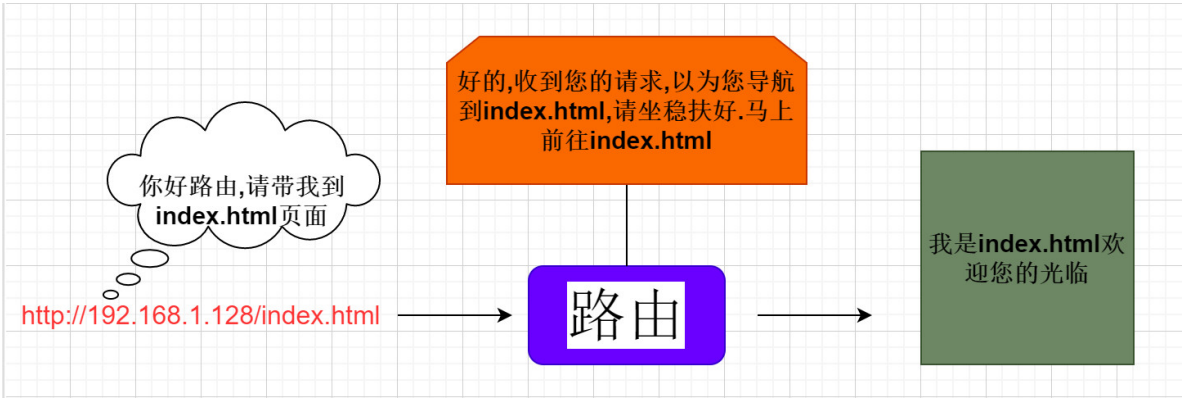
```
      beforeCreate() {
        alert("1.beforeCreate函数在组件实例化之前执行");
        alert(this.message); //undefined
        this.show(); // this.show is not a function
      },
      created() {
        alert("2.created函数执行时,组件实例化完成,但是DOM(页面)还未生成");
        alert(this.message);
        this.show();
      },
      beforeMount() {
        alert(
          "3.beforeMount函数执行时，模板已经在内存中编辑完成了，尚未被渲染到页面中"
        );
        alert(document.getElementById("msg").innerText); //Cannot read property
'innerText' of null
      },
      mounted() {
        alert("4.mounted函数执行时,模板已经渲染到页面,执行完页面显示");
        alert(document.getElementById("msg").innerText);
      },
      beforeUpdate() {
        alert("5.beforeUpdate执行时，内存中的数据已更新，但是页面尚未被渲染");
        alert("页面显示的内容:" + document.getElementById("msg").innerText);
        alert("data中的message数据是: " + this.message);
      },
      updated() {
        alert("6.updated执行时，内存中的数据已更新,此方法执行完显示页面!");
      },
    });
  </script>
```

1.9 Vue Router 路由

1.9.1 什么是路由?

在Web开发中，路由是指根据URL分配到对应的处理程序。 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图单页面web应用（single page web application，SPA）

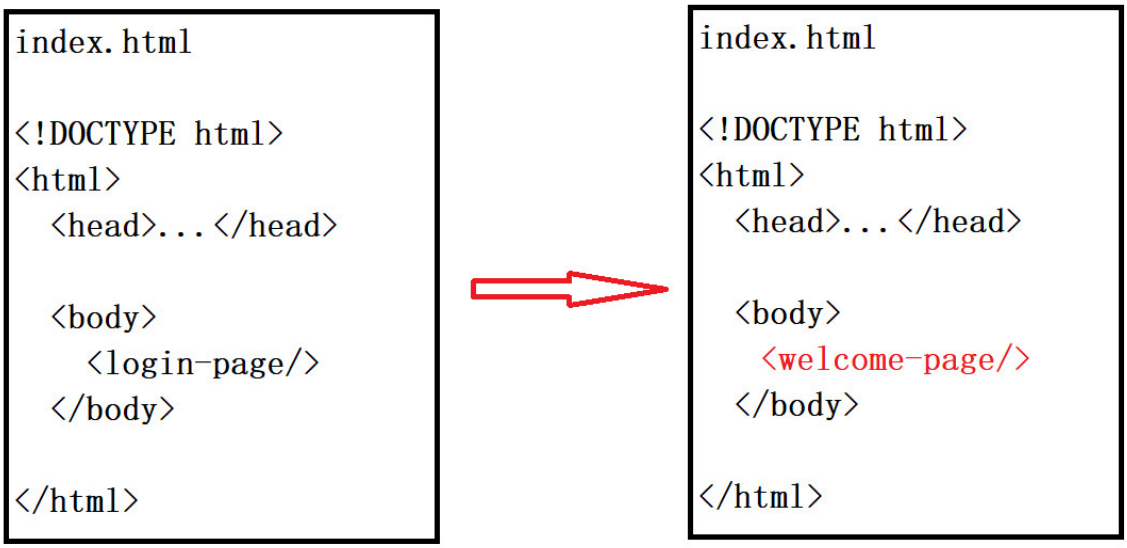


1.9.2 什么是SPA？

百度百科

- > 单页面Web应用（single page web application，SPA），就是只有一张Web页面的应用，是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。

单页应用不存在页面跳转，它本身只有一个HTML页面。我们传统意义上的页面跳转在单页应用的概念下转变为了`body`内某些元素的替换和更新，举个例子:



整个body的内容从登录组件变成了欢迎页组件, 从视觉上感受页面已经进行了跳转。但实际上，页面只是随着用户操作，实现了局部内容更新,依然还是在index.html 页面中。

单页面应用的好处:

1. 用户操作体验好，用户不用刷新页面，整个交互过程都是通过Ajax来操作。
2. 适合前后端分离开发，服务端提供http接口，前端请求http接口获取数据，使用JS进行客户端渲染。

1.9.3 路由相关的概念

- **router** ：

  是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用（SPA）变得易如反掌 ,router 就相当于一个管理者，它来管理路由。

- **route**:

  ruter相当于路由器, route就相当于一条路由.比如: Home按钮 => home内容， 这是一条route, news按钮 => news内容， 这是另一条路由。

- **routes** :

  是一组路由，把上面的每一条路由组合起来，形成一个数组。[{home 按钮 =>home内容 }， { about按钮 => about 内容}]

- **router-link组件**:

  router-link 是一个组件，是对标签的一个封装. 该组件用于设置一个导航链接，切换不同 HTML 内容。 **to** 属性为目标地址， 即要显示的内容

- **router-view 组件**:

  路由导航到指定组件后,进行渲染显示页面.

1.9.4 使用路由

1) Vue.js 路由需要载入 vue-router 库

```
//方式1: 本地导入
<script src="vue-router.min.js"></script>

//方式2: CDN
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
```

2) 使用步骤

1. 定义路由所需的组件
2. 定义路由 每个路由都由两部分 path (路径) 和component (组件)
3. 创建router路由器实例 ,管理路由
4. 创建Vue实例, 注入路由对象, 使用$mount() 指定挂载点

```
Vue 的$mount()为手动挂载，在项目中可用于延时挂载（例如在挂载之前要进行一些其他操作、判断等），之后要手动挂载上。new Vue时，el和$mount并没有本质上的不同。
```

3) HTM代码

```
<body>
    <div id="app">
      <h1>渣浪.com</h1>
      <p>
        <!-- 使用 router-link 组件来导航,to属性指定链接 -->
        <router-link to="/home">go to home</router-link>
        <router-link to="/news">go to news</router-link>
      </p>

      <!-- 路由的出口, 路由匹配到的组件(页面)将渲染在这里 -->
      <router-view></router-view>
    </div>
  </body>
```

4) JS代码

```
  <script src="./vue.min.js"></script>
  <script src="./vue-router.min.js"></script>
  <script>
    //1.定义路由所需的组件
    const home = { template: "<div>首页</div>" };
    const news = { template: "<div>新闻</div>" };

    //2.定义路由 每个路由都有两部分 path和component
    const routes = [
      { path: "/home", component: home },
      { path: "/news", component: news },
    ];

    //3.创建router路由器实例,对路由对象routes进行管理.
    const router = new VueRouter({
      routes: routes,
    });

    //4.创建Vue实例, 调用挂载mount函数,让整个应用都有路由功能
```

```
    const VM = new Vue({
      router,
    }).$mount("#app"); //$mount是手动挂载代替el
  </script>
```

1.9.5 路由总结

1. router是Vue中的路由管理器对象,用来管理路由.
2. route是路由对象,一个路由就对应了一条访问路径,一组路由用routes表示
3. 每个路由对象都有两部分 path(路径)和component (组件)
4. router-link 是对a标签的封装,通过to属性指定连接
5. router-view 路由访问到指定组件后,进行页面展示

###