一、VUE简介

1.1 使用jQuery的复杂性问题

- 使用jQuery进行前后端分离开发，既可以实现前后端交互（ajax）,又可以完成数据渲染；
- 存在的问题：jQuery需要通过HTML标签拼接、DOM节点操作完成数据的显示，开发效率低且容易出错，渲染效率较低
- vue 是继jQuery之后的又一优秀的前端框架：专注于前端数据的渲染——语法简单、渲染效率高

1.2 VUE简介

1.2.1 前端框架

- 前端三要素：HTML、CSS、JavaScript
  - HTML决定网页结构
  - CSS决定显示效率
  - JavaScript决定网页功能（交互、数据显示）
- UI框架：
  - Bootstrap
  - amazeUI
  - Layui
- JS框架：
  - jQuery（jQuery UI）
  - React
  - angular
  - nodejs----后端开发
  - vue 集各种前端框架的优势发展而来

1.2.2 MVVM

> 项目结构经历的三个阶段：

后端MVC 我们就可以理解为单体架构，流程控制是由后端控制器来完成

前端MVC 前后端分离开发，后端只负责接收响应请求

MVVM 前端请求后端接口，后端返回数据，前端接收数据，并将接收的数据设置“VM”,HTML从vm取值

- M model 数据模型 指的是后端接口返回的数据
- V view 视图
- VM ViewModel 视图模型 数据模型与视图之间的桥梁，后端返回的model转换前端所需的vm，视图层可以直接从vm中提取数据

MVC	MVVM

二、 vue的入门使用

> 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.

2.1 vue的引入

- 离线引用：下载vue的js文件，添加到前端项目，在网页中通过script标签引用vue.js文件
- CDN引用：

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

2.2 入门案例

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <script src="js/vue.js"></script>
  </head>
  <body>
    <div id="container">
      从vm中获取的数据为：{{str}}
    </div>

    <script type="text/javascript">
      var vm = new Vue({
```

```
        el:"#container",
        data:{
          str:"从前有座山"
        }
      });
    </script>
  </body>
</html>
```

三、 vue的语法

3.1 基本类型数据和字符串

```
{{code}}
{{str}}
-------------------------
data:{
code:10,
str:"test"
}
```

3.2 对象类型数据

- 支持ognl语法

```
{{stu.stuNum}}
{{stu.stuName}}
-----------------------------
data{
    stu:{
        stuNum:"100001",
        stuName:"张三",
        stuGender:"M",
        stuAge:20
    }
}
```

3.3 条件 v-if

```
<label v-if="stu.stuGender=='M'">男</label><br/>
-----------------------------------------------------
data:{
    stu:{
        stuNum:"100001",
        stuName:"张三",
        stuGender:"M",
        stuAge:20
    }
}
```

3.4 循环 v-for

```
<table border="1" cellspacing="0" width="400">
    <tr>
        <th>序号</th>
        <th>学号</th>
        <th>姓名</th>
        <th>性别</th>
        <th>年龄</th>
    </tr>
    <tr v-for="s,index in stus">
        <td>{{index+1}}</td>
        <td>{{s.stuNum}}</td>
        <td>{{s.stuName}}</td>
        <td>
            <label v-if="s.stuGender == 'M'">男</label>
            <label v-if="s.stuGender == 'F'">女</label>
        </td>
        <td>{{s.stuAge}}</td>
    </tr>
</table>

-----------------------------------------------------------------------
data:{
    stus:[
        {
            stuNum:"100001",
            stuName:"张大三",
            stuGender:"M",
```

```
            stuAge:23
        },
        {
            stuNum:"100002",
            stuName:"张中三",
            stuGender:"M",
            stuAge:22
        },
        {
            stuNum:"100003",
            stuName:"张小三",
            stuGender:"F",
            stuAge:20
        }
    ]
}
```

3.5 v-bind绑定标签属性

- `v-bind:` 可简写为 `:`

```
<input type="text" v-bind:value="str"/>
<img :src="stu.stuImg"/>
-----------------------------------
data{
  str:"从前有座山",
  stu:{
        stuImg:"img/01.jpg"
    }
}
```

3.6 表单标签的双向绑定 v-model

- 只能使用在表单输入标签
- v-model:value 可以简写为 v-model

```
<input type="text" v-model:value="str"/>
<input type="text" v-model="str"/>
-----------------------------------
data{
  str:"从前有座山"
}
```

四、vue实例

> 每个使用vue进行数据渲染的网页文档都需要创建一个Vue实例 —— ViewModel

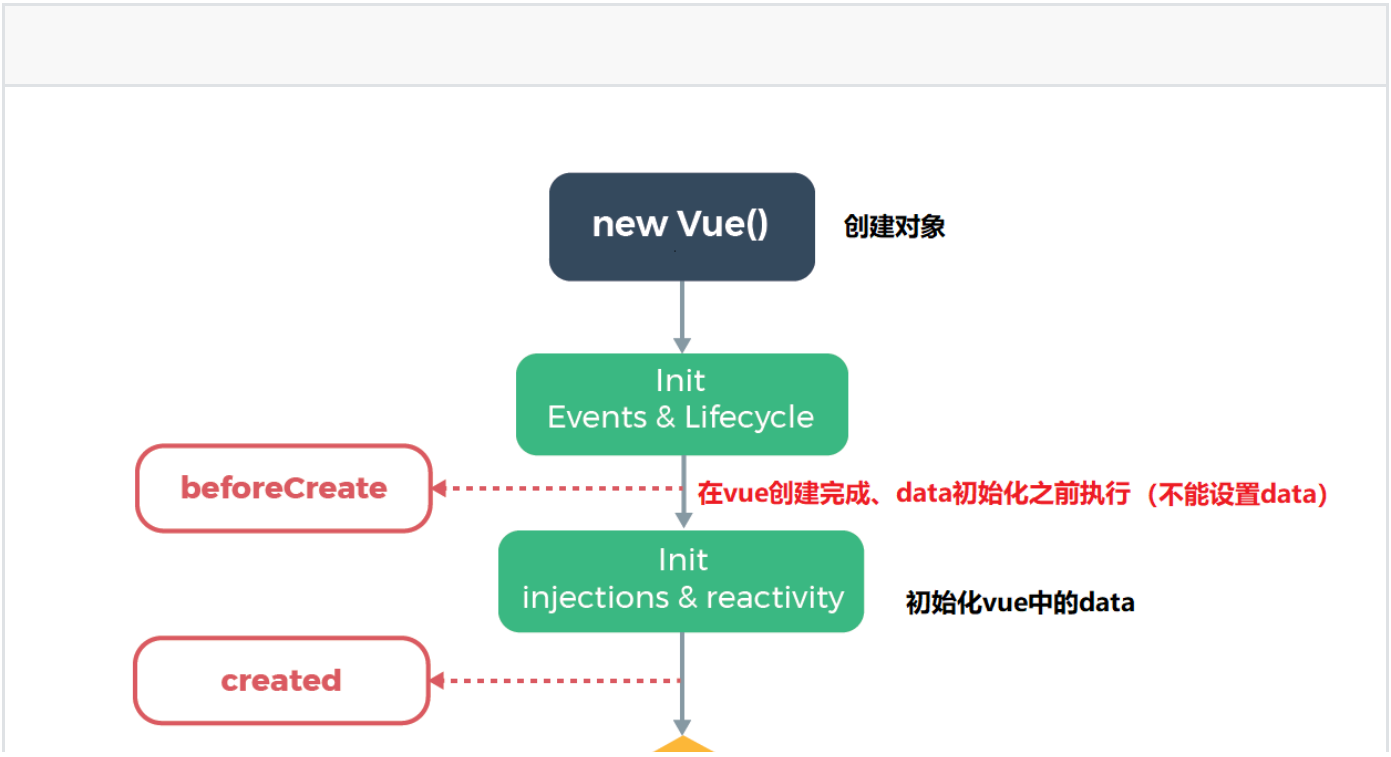
4.1 Vue实例的生命周期

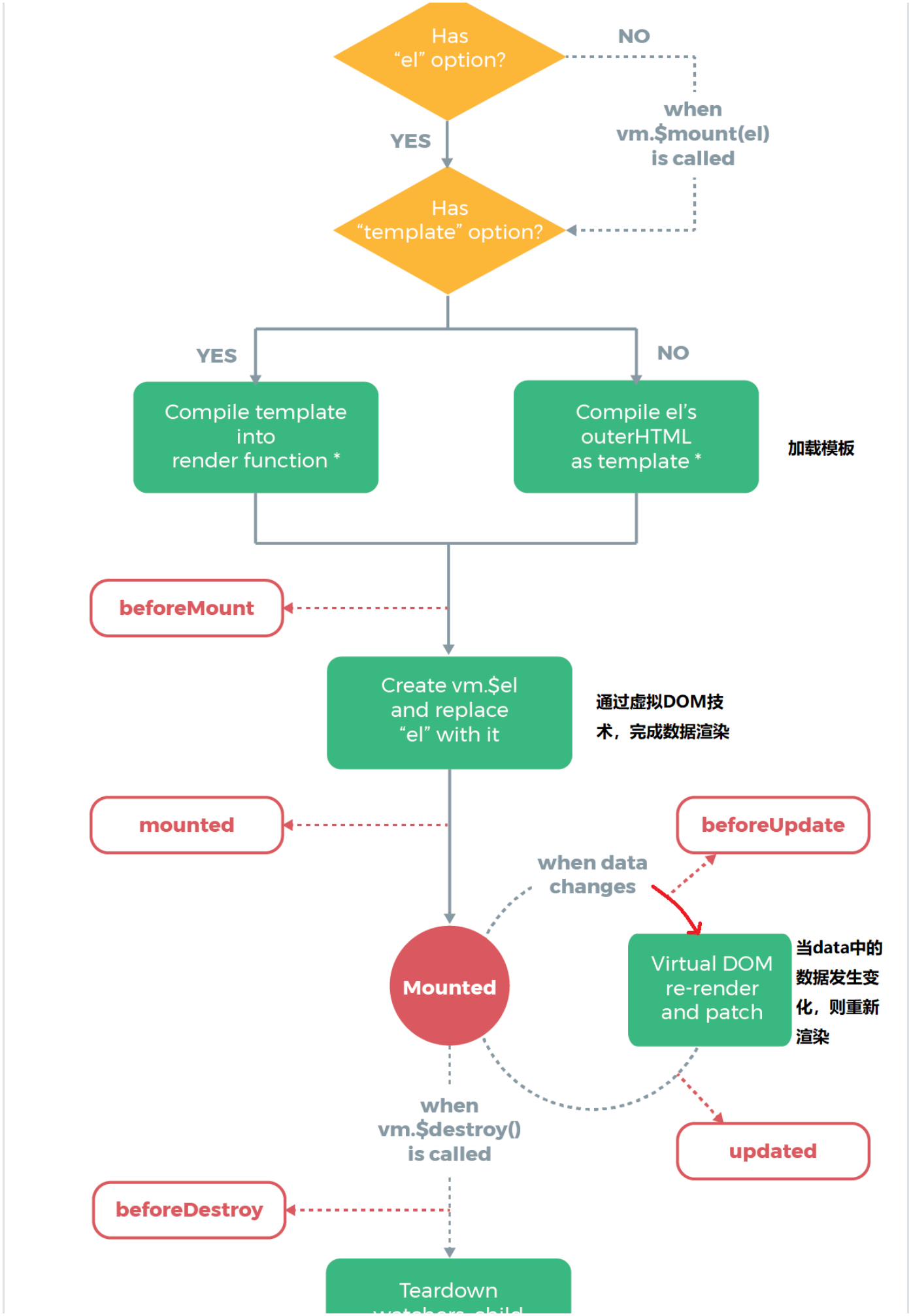
> vue实例生命周期——vue实例从创建到销毁的过程

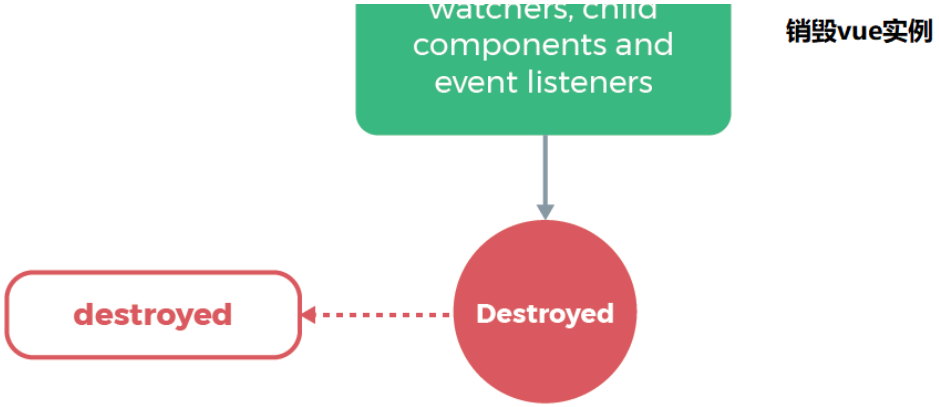
- 创建vue实例（初始化data、加载el）
- 数据挂载（将vue实例data中的数据渲染到网页HTML标签）
- 重新渲染（当vue的data数据发生变化，会重新渲染到HTML标签）
- 销毁实例

4.2 钩子函数

> 为了便于开发者在vue实例生命周期的不同阶段进行特定的操作，vue在生命周期四个阶段的前后分别提供了一个函数，这个函数无需开发者调用，当vue实例到达生命周期的指定阶段会自动调用对应的函数。




Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *
加载模板
beforeMount
Create vm.$el and replace "el" with it
通过虚拟DOM技术，完成数据渲染
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
当data中的数据发生变化，则重新渲染
when vm.$destroy() is called
updated
beforeDestroy
Teardown



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

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <label v-once>{{str}}</label><br/>
      <label>{{str}}</label><br/>
      <input type="text" v-model="str"/>
    </div>

    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{},
        beforeCreate:function(){
          //1.data初始化之前执行，不能操作data
        },
        created:function(){
          //2.data初始化之后执行，模板加载之前，可以修改/获取data中的值
          console.log(this.str);
          //this.str = "山里有座庙";
        },
```

```
        beforeMount:function(){
          //3.模板加载之后，数据初始渲染（挂载）之前，可以修改/获取data中的值
          //this.str = "庙里有口井";
        },
        mounted:function(){
          //4.数据初始渲染（挂载）之后，可以对data中的变量进行修改，但是不会影响v-once的渲染
          //this.str = "井里有只蛙";
        },
        beforeUpdate:function(){
          //5.数据渲染之后，当data中的数据发生变化触发重新渲染，渲染之前执行此函数
          //  data数据被修改之后，重新渲染到页面之前
          console.log("-----"+this.str);
          this.str = "从前有座山2";
        },
        updated:function(){
          //6.data数据被修改之后，重新渲染到页面之后
          //this.str = "从前有座山3";
        },
        beforeDestroy:function(){
          //7.实例销毁之前
        },
        destroyed:function(){
          //8.实例销毁之后
        }

      });

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

五、计算属性和侦听器

5.1 计算属性

> data中的属性可以通过声明获得，也可以通过在computed通过计算获得
>
> 特性：计算属性所依赖的属性值发生变化会影响计算属性的值同时发生变化

示例

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <input type="text" v-model="str1"/><br/>
      <input type="text" v-model="str2"/><br/>
      {{str3}}
    </div>

    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{
          str1:"千锋",
          str2:"武汉"
        },
        computed:{
          str3:function(){
            return this.str1+this.str2;
          }
        }
      });
    </script>
  </body>
</html>
```

5.2 侦听器

> 侦听器，就是data中属性的监听器，当data中的属性值发生变化就会触发侦听器函数的执行

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <script type="text/javascript" src="js/vue.js" ></script>
```

```
  </head>
  <body>
    <div id="container">
      <input type="text" v-model="str1"/><br/>
      <input type="text" v-model="str2"/><br/>
      {{str3}}
    </div>

    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{
          str1:"千锋",
          str2:"武汉",
          str3:"千锋武汉"
        },
        watch:{
          str1:function(){
            this.str3 = this.str1 +this.str2;
          },
          str2:function(){
            this.str3 = this.str1 +this.str2;
          }
        }
      });
    </script>
  </body>
</html>
```

六、class与style绑定

> 我们可以使用mustache语法将vue中data的数据绑定到HTML标签及标签的属性，如何将data中的值绑定到标签的class及style属性呢?

6.1 class绑定

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

```
      .mystyle1{
        width: 200px;
        height: 100px;
        background: orange;
      }
      .mystyle3{
        width: 200px;
        height: 100px;
        background: black;
      }
      .my-style2{
        border-radius: 10px;
      }
    </style>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <!--如果b1为true就加载 mystyle1；如果b2为true，则加载my-style2-->
      <div :class="{mystyle1:b1,'my-style2':b2}"></div>

      <!--为class属性加载多个样式名 -->
      <div :class="[chooseStyle1,chooseStyle2]"></div>

      <!--如果b3为true，则class='mystyle3'; 否则class='mystyle1'
      如果在三目运算中使用样式名则需加单引号，不加单引号则表示从data变量中获取样式名-->
      <div :class="[b3 ? 'mystyle3' : 'mystyle1']"></div>
      <div :class="[b3 ? chooseStyle3 : chooseStyle1]"></div>

    </div>
    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{
          b1:true,
          b2:true,
          b3:false,
          chooseStyle1:"mystyle1",
          chooseStyle2:"my-style2",
          chooseStyle3:"mystyle3"
        }
```

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

```

6.2 style绑定

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

    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <!--当使用v-bind绑定内联样式时:
        1. 使用{}定义style样式，才能获取data中的值，{}要遵循JSON格式
        2. {}中不在使用style样式属性名“font-size”,而要使用对应的js属性名
        border-style-width ---  borderStyleWidth
      -->
      <div v-bind:style="{color: colorname,fontSize: fontsize+'px'
}">WH2010</div>

      <!--我们可以直接为style属性绑定一个data中定义好的内联样式的字符串-->
      <div v-bind:style="mystyle1">千锋Java-WH2010</div>

      <!--我们可以直接为style属性绑定一个data中定义好的内联样式的对象-->
      <div v-bind:style="mystyle2">千锋Java-WH2010</div>

      <!--可以在同一个style上通过数组引用多个内联样式的对象-->
      <div v-bind:style="[mystyle2,mystyle3]">千锋Java-WH2010</div>
    </div>

    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{
          colorname:"green",
          fontsize:30,
```

```
            mystyle1:"color:orange;font-size:45px",
            mystyle2:{
              color:"blue",
              fontSize:"40px"
            },
            mystyle3:{
              textShadow:"orange 3px 3px 5px"
            }
          }
        });
    </script>
  </body>
</html>
```

七、条件与列表渲染

7.1 条件渲染

7.1.1 v-if

> 在html标签可以添加v-if指令指定一个条件，如果条件成立则显示此HTML标签，如果不成立则不显示当前标签；
>
> 条件可以是一个表达式也可以是一个具体的bool类型值

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

    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">

      <h3 v-if="b">从前有座山</h3>
      <h3 v-if="code == 1">从前有座山</h3>
    </div>

    <script type="text/javascript">
      var vm = new Vue({
```

```
        el:"#container",
        data:{
          code:2,
          b:false
        }
      });
    </script>
  </body>
</html>
```

7.1.2 v-else

```
<div id="container">
    <!--v-else标签需要紧跟在v-if的标签之后，中间不能有其他标签-->
    <h3 v-if="code == 1">从前有座山</h3>
    <h3 v-else>山里有座庙</h3>
</div>

<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        data:{
            code:1
        }
    });
</script>
```

7.1.3 v-else-if

```
<div id="container">
    <h3 v-if="code >= 90">优秀</h3>
    <h3 v-else-if="code >= 80">良好</h3>
    <h3 v-else-if="code >= 70">中等</h3>
    <h3 v-else-if="code >= 60">及格</h3>
    <h3 v-else>不想理你</h3>
</div>

<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        data:{
            code:85
```

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

7.1.4 v-show

> 从功能上将v-show和v-if作用是相同的，渲染过程有区别

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，`v-if` 有更高的切换开销，而 `v-show` 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好。

7.2 列表渲染

> 将集合数据以表格、列表的形式显示

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <link rel="stylesheet" href="css/bootstrap.css" />
    <script type="text/javascript" src="js/jquery-3.4.1.min.js" >
</script>
    <script type="text/javascript" src="js/bootstrap.js" ></script>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <ul>
        <li v-for="c in categories">
          <a :href="'query?cid='+c.cid">{{c.cname}}</a>
        </li>
      </ul>

```

```
      <table class="table table-bordered">
        <tr>
          <th>学号</th>
          <th>照片</th>
          <th>姓名</th>
          <th>性别</th>
          <th>年龄</th>
          <th>操作</th>
        </tr>
        <template v-for="s,index in stus">
          <tr :id="'tr'+s.stuNum">
            <td>{{s.stuNum}}</td>
            <td>
              <img height="30" :src="s.stuImg"/>
            </td>
            <td>{{s.stuName}}</td>
            <td>
              <!--{{s.stuGender=='M'?'男':'女'}}-->
              <img v-if="s.stuGender=='M'" src="img/m.bmp">
              <img v-else src="img/f.bmp">
            </td>
            <td>{{s.stuAge}}</td>
            <td>
              <a class="btn btn-danger btn-xs" :href="'stu/delete?cid='+s.stuNum">删除</a>
              <a class="btn btn-success btn-xs" :href="'stu/update?cid='+s.stuNum">修改</a>
            </td>
          </tr>
        </template>
      </table>
    </div>

    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        data:{
          categories:[
            {
              cid:1,
              cname:"华为"
            },
```

```
            {
              cid:2,
              cname:"小米"
            },
            {
              cid:3,
              cname:"OPPO"
            },
            {
              cid:4,
              cname:"VIVO"
            }
          ],
          stus:[
            {
              stuNum:"10010",
              stuImg:"img/01.jpg",
              stuName:"Tom",
              stuGender:"M",
              stuAge:20
            },
            {
              stuNum:"10011",
              stuImg:"img/02.jpg",
              stuName:"LiLei",
              stuGender:"M",
              stuAge:20
            },
            {
              stuNum:"10012",
              stuImg:"img/03.jpg",
              stuName:"Lucy",
              stuGender:"F",
              stuAge:20
            },
            {
              stuNum:"10013",
              stuImg:"img/04.jpg",
              stuName:"Polly",
              stuGender:"F",
              stuAge:20
            }
```

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

八、事件处理

- 在使用vue进行数据渲染时，如果使用原生js事件绑定(例如onclick)，如果需要获取vue实例中的数据并传参则需要通过拼接来完成
- vue提供了v-on指令用于绑定各种事件（v-on:click），简化了从vue取值的过程，但是触发的方法需要定义在vue实例的 methods中

```
<button type="button"  v-on:click="doDelete(s.stuNum,s.stuName)">删除</button>

<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        data:{},
        methods:{
            doDelete:function(snum,sname){
                console.log("----delete:"+snum+"    "+sname)
            }
        }
    });
</script>
```

- `v-on:clicl`可以缩写为`@click`

8.1 使用JS函数传值

```
<button type="button" class="btn btn-danger btn-xs" v-on:click="doDelete(s.stuNum,s.stuName)">删除</button>

<script>
    var vm = new Vue({
        el:"#container",
        data:{};
        methods:{
            doDelete:function(snum,sname){
                console.log("----delete:"+snum+"    "+sname)
            }
        }
    });
</script>
```

8.2 使用dataset对象传值

```
<button type="button" class="btn btn-success btn-xs" @click="doUpdate" :data-snum="s.stuNum"
                :data-sname="s.stuName" :data-simg="s.stuImg">修改</button>
<script>
    var vm = new Vue({
        el:"#container",
        data:{};
        methods:{
            doUpdate:function(event){
                //如果v-on绑定的js函数没有参数，调用的时候可以省略()，同时可以给js函数一个event参数(事件对象)
                // 1. event 表示触发当前函数的事件
                // 2. event.srcElement 表示发生事件的元素---修改按钮
                // 3. event.srcElement.dataset 表示按钮上绑定的数据集(data-开头的属性)
                console.log("-----update")
                var stu = event.srcElement.dataset;
            }
        }
    });
</script>
```

8.3 混合使用

- `$event`

```
<button type="button" class="btn btn-danger btn-xs" v-on:click="doDelete(s.stuNum,s.stuName,$event)":data-simg="s.stuImg">删除</button>

<script>
    var vm = new Vue({
        el:"#container",
        data:{};
        methods:{
            doDelete:function(snum,sname,event){
                console.log("----delete:"+snum+"    "+sname)
                console.log(event.srcElement.dataset);
            }
        }
    });
</script>
```

8.4 事件修饰符

> 当使用v-on进行事件绑定的时候，可以添加特定后缀，设置事件触发的特性

8.4.1 事件修饰符使用示例

```
<button type="submit" @click.prevent="事件函数">测试</button>
```

8.4.2 事件修饰符

.prevent 消除元素的默认事件

```
<div id="container">
    <form action="https://www.baidu.com">
        <button type="submit" class="btn btn-success btn-xs" @click.prevent="test">测试</button>
    </form>
</div>

<script type="text/javascript">
    var vm = new Vue({
```

```
        el:"#container",
        data:{

        },
        methods:{
            test:function(){
                console.log("---test");
            }
        }
    });
</script>
```

.stop 阻止事件冒泡（阻止子标签向上冒泡）

.self 设置只能自己触发事件（子标签不能触发）

```
<div id="container">
    <div style="width: 200px; height: 200px; background: red;" @click.self="method1">
        <div style="width: 150px; height: 150px; background: green;" @click="method2">
            <button type="button" @click.stop="method3">测试</button>
        </div>
    </div>
</div>

<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        data:{

        },
        methods:{
            method1:function(){
                alert("1");
            },
            method2:function(){
                alert("2");
            },
            method3:function(){
                alert("3");
            }
```

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

.once 限定事件只触发一次

8.4.3 按键修饰符

> 按键修饰符就是针对键盘事件的修饰符，限定哪个按键会触发事件

```
<input type="text" @keyup.enter="method4"/>
```

- `.enter`
- `.tab`
- `.delete` (捕获“删除”和“退格”键)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

除了以上vue提供按钮的别名之外，我们还可以根据键盘为按键自定义别名

[键盘码]

示例:

```
<div id="container">
   <!--2.使用自定义的按键别名aaa作为修饰符-->
    <input type="text" @keyup.aaa="method4"/>
</div>
<script type="text/javascript">
    //1.为按键J定于别名为  aaa
    Vue.config.keyCodes.aaa =74;

    var vm = new Vue({
        el:"#container",
        data:{},
        methods:{
            method4:function(){
                alert("4");
```

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

8.4.3 系统修饰符

> 组合键

示例 ctrl+j触发事件

```
<div id="container">
    <input type="text" @keyup.ctrl.j="method4"/>
</div>
<script type="text/javascript">
    Vue.config.keyCodes.j =74;

    var vm = new Vue({
        el:"#container",
        data:{},
        methods:{
            method4:function(){
                alert("4");
            }
        }
    });
</script>
```

- `.ctrl`
- `.alt`
- `.shift`
- `.meta` windows键

九、表单输入绑定

> 表单输入绑定，即双向绑定：就是能够将vue实例的data数据渲染到表单输入视图（input\textarea\select），也能够将输入视图的数据同步更新到vue实例的data中

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
```

```
    <title></title>
    <link rel="stylesheet" href="css/bootstrap.css" />
    <script type="text/javascript" src="js/jquery-3.4.1.min.js" >
</script>
    <script type="text/javascript" src="js/bootstrap.js" ></script>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <!--文本输入框、密码输入框-->
      <input type="text" v-model="text"/><br/>
      <input type="password" v-model="pwd"/><br/>

      <!--单选按钮-->
      <input type="radio" v-model="opt1" value="A"/>A  3
      <input type="radio" v-model="opt1" value="B"/>B  4
      <input type="radio" v-model="opt1" value="C"/>C  5
      <input type="radio" v-model="opt1" value="D"/>D  6  <br/>

      <!--复选框，绑定的是一个数组-->
      <input type="checkbox" v-model="opt2" value="篮球"/>篮球 <br/>
      <input type="checkbox" v-model="opt2" value="足球"/>足球 <br/>
      <input type="checkbox" v-model="opt2" value="羽毛球"/>羽毛球 <br/>
      <input type="checkbox" v-model="opt2" value="乒乓球"/>乒乓球<br/>

      <!--下拉菜单select：绑定一个字符串-->
      <select v-model="city">
        <option value="BJ">北京</option>
        <option value="SH">上海</option>
        <option value="GZ">广州</option>
        <option value="SZ">深圳</option>
      </select>
      <br/>
      <!--下拉菜单select：如果有multiple表示可多选，需要绑定一个数组-->
      <select v-model="cities" multiple>
        <option value="BJ">北京</option>
        <option value="SH">上海</option>
        <option value="GZ">广州</option>
        <option value="SZ">深圳</option>
      </select>
      <br/>
      <button type="button" @click="doSearch">测试</button>
```

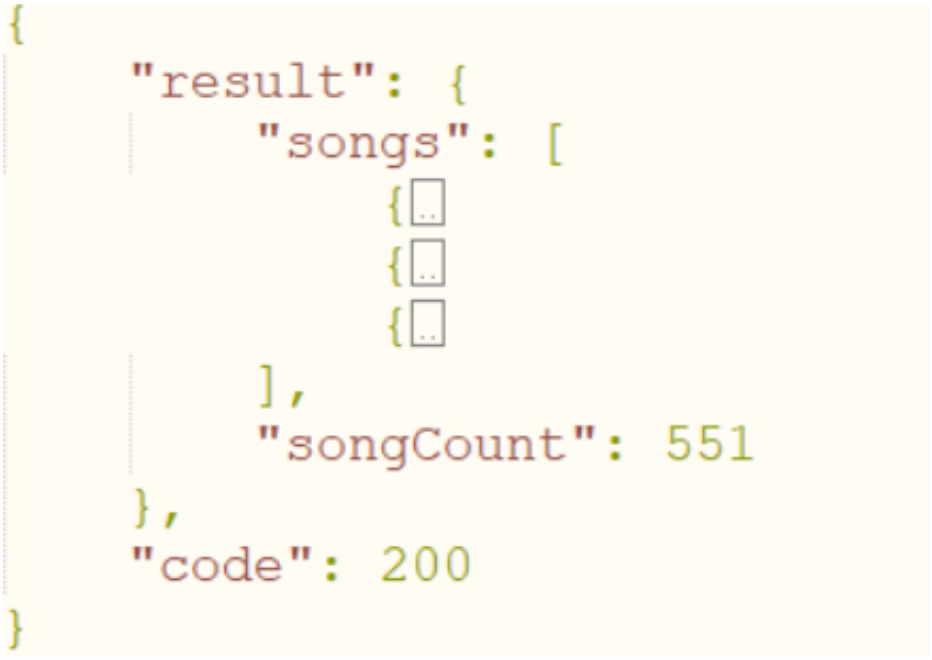
```
    </div>

    <script type="text/javascript">

      var vm = new Vue({
        el:"#container",
        data:{
          text:"aaa",
          pwd:"111111",
          opt1:"C",
          opt2:["篮球","羽毛球"],
          city:"SZ",
          cities:["BJ","SZ"]
        },
        methods:{
          doSearch:function(){
            alert(vm.cities);
          }
        }
      });
    </script>
  </body>
</html>
```

十、vue使用案例

10.1 接口说明

接口名称	
功能描述	根据关键字搜索音乐信息
请求URL	http://47.96.11.185:9999/music/search
请求方式	GET | POST
请求参数	s string [必须] 搜索关键字 limit int [可选] 返回的搜索结果的条数，默认为10 type int [可选] 搜索类型(1单曲 10歌单)，默认为1 offset int [可选] 搜索结果的偏移
返回结果	

10.2 如何部署jar文件

```
java -jar music-1.0.0.jar
```

10.3 案例目标

请大家根据以上接口实现搜索和列表显示功能

10.4 案例实现

10.4.1 音乐搜索

10.4.2 音乐播放

- 在music.html中定义音频播放器(定义在vue的容器之外)

```
<audio controls style="width:100%" src="" id="player"></audio>
```

- 给播放按钮绑定点击事件触发的函数doPlay

```
<button type="button" class="btn btn-success btn-xs" @click="doPlay"
:data-mid="song.id">播放</button>
```

- 在doPlay中执行播放

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

    var player = document.getElementById("player");

var vm = new Vue({
    el:"#container",
    data:{
        keyword:"张韶涵",
        songs:[],
        currentid:0
    },
    methods:{
        doSearch:function(){
            console.log(vm.keyword);
            $.get("http://localhost:9999/music/search",
{s:vm.keyword,limit:15,offset:0},function(res){
                console.log(res);
                if(res.code==200){
                    //获取此关键词搜索的总记录数
                    var count = res.result.songCount;
                    //获取音乐集合
                    var arr = res.result.songs;
                    vm.songs = arr;
                }else{
                    vm.songs = data;
                }
            },"json");
```

```
        },
        doPlay:function(event){
            vm.currentid = event.srcElement.dataset.mid;
            //网易云音乐播放地址:http://music.163.com/song/media/outer/url?id=songId
            player.src = "http://music.163.com/song/media/outer/url?id="+vm.currentid;
            player.play();
        }
    }
});

</script>
```

10.4.3 播放暂停切换

十一、组件

11.1 组件介绍及示例

> 组件，就是将通用的HTML模块进行封装——可复用

11.1.1 组件注册

> 将通用的HTML模块封装注册到vue中

```
Vue.component("header-bar",{

});
```

11.1.2 组件引用

- 定义组件需要依赖vue.js，在引用自定义组件的js文件之前要先引用vue.js
- 组件的引用必须在vue实例el指定的容器中

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

```

```
  </head>
  <body>
    <div id="container">
      <header-bar></header-bar>
    </div>
    <script type="text/javascript" src="js/vue.js" ></script>
    <script type="text/javascript" src="js/my-components.js" ></script>
    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
      });
    </script>
  </body>
</html>

```

11.2 组件注册

11.2.1 自定义组件的结构

- `data` 定义组件的模板渲染的数据
- `template` 组件的HTML模块（HTML标签\css样式）
- `methods` 定义组件中的标签事件绑定的JS函数

```
Vue.component("header-bar",{
  data:function(){
    //组件中的data是通过函数返回的对象
    return {
      title:"Java2010电商平台"
    };
  },
  template:`<div style="width: 100%; height: 80px; background:
lightyellow;">
        <table width="100%">
          <tr>
            <td width="200" align="right" valign="middle">
              <img src="img/logo.png" height="80">
            </td>
            <td>
              <label style="color: deepskyblue;font-size:32px; font-
family: 华文行楷; margin-left: 30px;">
                {{title}}

```

```
              </label>
            </td>
            <td>
              <button @click="test">组件中的按钮</button>
            </td>
          </tr>
        </table>
      </div>`,
  methods:{
    test:function(){
      alert("组件中定义的函数");
    }
  }
});
```

11.2.2 组件的封装

- 将模版中的css样式提出取来，单独定义到css文件存储在css目录
- 将模版中的图片存在在img目录
- 将定义组件的js文件和vue的文件存放到js目录

vue组件封装的目录结构

11.2.3 组件的复用

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
    <link rel="stylesheet" href="css/my-components.css" />
  </head>
  <body>
    <div id="container">
      <header-bar></header-bar>
```

```
    </div>
    <script type="text/javascript" src="js/vue.js" ></script>
    <script type="text/javascript" src="js/my-components.js" ></script>
    <script type="text/javascript">
      var vm = new Vue({
        el:"#container"
      });
    </script>
  </body>
</html>
```

11.3 组件通信

> vue实例本身就是一个组件（模板就是el指定容器 ,data就是组件数据,methods就是组件的事件函数）
> 在vue实例指定的el容器中引用的组件称为子组件 ,当前vue实例就是父组件

11.3.1 父传子

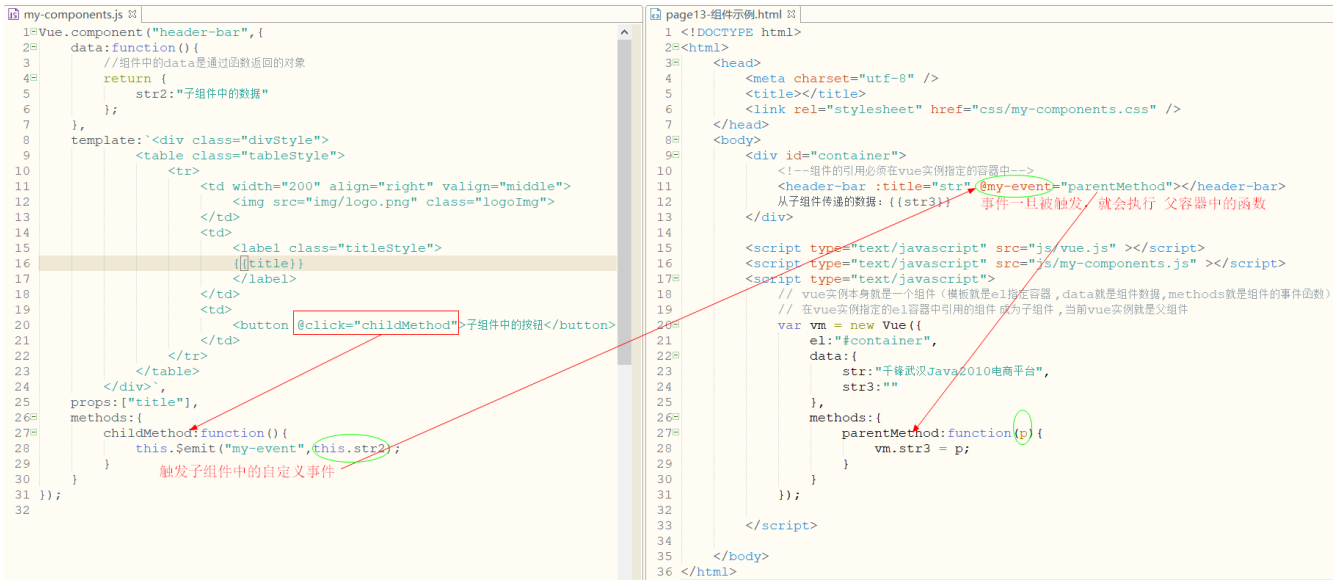
> vue实例引用组件的时候，传递数据到引用的组件中

示意图

11.3.2 子传父

> 通过子组件的按钮“调用”父组件的函数，通过函数传值

调用流程示意图



11.4 组件插槽

> 当我们自定义vue组件时，允许组件中的部分内容在调用组件时进行定义——插槽

11.4.1 插槽的使用

- 在自定义组件时通过`slot`标签在组件的模版中定义插槽

```
Vue.component("header-bar",{
  data:function(){
    //组件中的data是通过函数返回的对象
    return {
      str2:"子组件中的数据"
    };
  },
  template:`<div class="divStyle">
      <table class="tableStyle">
        <tr>
          <td width="200" align="right" valign="middle">
            <img src="img/logo.png" class="logoImg">
          </td>
          <td>
            <label class="titleStyle">
            {{title}}
            </label>
          </td>
          <td>
```

```
            <slot></slot>
          </td>
          <td>
            <button @click="childMethod">子组件中的按钮</button>
          </td>
        </tr>
      </table>
    </div>`,
  props:["title"],
  methods:{
    childMethod:function(){
      this.$emit("my-event",this.str2);
    }
  }
});
```

- 在父组件中调用此组件时，指定插槽填充的模版

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <link rel="stylesheet" href="css/bootstrap.css" />
    <link rel="stylesheet" href="css/my-components.css" />
    <script type="text/javascript" src="js/jquery-3.4.1.min.js" >
</script>
    <script type="text/javascript" src="js/bootstrap.js" ></script>
    <script type="text/javascript" src="js/vue.js" ></script>
  </head>
  <body>
    <div id="container">
      <header-bar :title="sss">
                <!--组件标签包含的HTML默认为填充到插槽的模版-->
        <input/><button>搜索</button>
      </header-bar>
    </div>

    <script type="text/javascript" src="js/my-components.js" >
</script>
    <script type="text/javascript">
      var vm = new Vue({
```

```
        el:"#container",
        data:{
          sss:"自定义标题"
        }
      });
    </script>
  </body>
</html>

```

11.4.2 具名插槽

> 当组件中的插槽数量>1时，需要给组件中的slot标签添加name属性指定插槽的名字

- 定义组件：

```
Vue.component("page-frame",{
  template:`<div>
        <div id="header" style="width:100%;
height:100px;background:pink">
          <slot name="s1"></slot>
        </div>
        <div style="width:100%; height:580px">
          <slot name="s2"></slot>
        </div>
    <div id="footer" style="width:100%;
height:40px;background:lightgray">{{cr}}</div>
      </div>`,
  props:["title","cr"]
});
```

- 引用组件 `template`

```
<div id="container">

    <page-frame title="标题" cr="千锋武汉">
        <!--定义一个模版，填充到组件的name=s1的 插槽-->
        <template slot="s1">
            <input type="text" placeholder="歌曲名、歌手" />
            <button type="button" @click="doSearch">搜索</button>
        </template>
        <!--定义一个模版，填充到组件的name=s2的 插槽-->
```

```
        <template slot="s2">
            <table class="table table-bordered table-condensed">
                <tr>
                    <th>序号</th>
                    <th>歌曲ID</th>
                    <th>歌曲名</th>
                    <th>歌手</th>
                    <th>专辑</th>
                    <th>时长</th>
                    <th>操作</th>
                </tr>
            </table>
        </template>
    </page-frame>

</div>
```

11.4.3 插槽作用域

- 定义组件时，将组件中的数据绑定到`slot`标签

```
Vue.component("page-frame",{
  template:`<div>
        <div id="header" style="width:100%;
height:100px;background:pink">
          <slot name="s1"></slot>
        </div>
        <div style="width:100%; height:580px">
          <slot name="s2" v-bind:musics="songs"></slot>
        </div>
        <div id="footer" style="width:100%;
height:40px;background:lightgray">{{cr}}</div>
      </div>`,
  props:["title","cr"],
  data:function(){
    return {
      songs:[
        {},{}
      ]
    };
  }
});
```

- 引用组件时，在填充插槽的模版上使用`slot-scopt`属性获取插槽绑定的值

```
<page-frame title="标题" cr="千锋武汉">
    <template slot="s1">
        <input type="text" placeholder="歌曲名、歌手" />
        <button type="button" @click="doSearch">搜索</button>
    </template>
    <!--在使用模版填充组件插槽时，可以使用slot-scope属性获取组件插槽绑定的数据的集合 -->
    <template slot="s2" slot-scope="res">
        <table class="table table-bordered table-condensed">
            <tr>
                <th>序号</th>
                <th>歌曲ID</th>
                <th>歌曲名</th>
                <th>歌手</th>
                <th>专辑</th>
                <th>时长</th>
                <th>操作</th>
            </tr>
            <tr v-for="song,index in res.musics">
                <td>{{index+1}}</td>
                <td>{{song.id}}</td>
                <td>
                    {{song.name}}

                </td>
                <td>
                    <span v-for="artist in song.artists">
                         {{artist.name}}
                    </span>
                </td>
                <td>{{song.album.name}}</td>
                <td width="8%">
                    {{   Math.floor( Math.round(song.duration/1000)/60) < 10 ? '0'+Math.floor( Math.round(song.duration/1000)/60) : Math.floor( Math.round(song.duration/1000)/60)    }}

     :
```

```
34
     {{   Math.round(song.duration/1000)%60 <10 ? '0'+(
Math.round(song.duration/1000)%60 ) : Math.round(song.duration/1000)%60
}}
35                  </td>
36                  <td width="10%">
37                      <button type="button"  class="btn btn-primary btn-
xs">播放</button>
38                  </td>
39              </tr>
40          </table>
41      </template>
42  </page-frame>
```

十二、axios

12.1 axios介绍

> vue可以实现数据的渲染，但是如何获取数据呢?
>
> vue本身不具备通信能力，通常结合axios——一个专注于异步通信的js框架来使用
>
> - axios 数据通信
> - vue 数据渲染

12.2 axios入门使用

- 原生ajax --- 实现步骤复杂
- jQuery 笨重
- axios 简洁、高效，对RESTful支持良好

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title></title>
6      <script type="text/javascript" src="js/vue.js" ></script>
7      <script type="text/javascript" src="js/axios.min.js" ></script>
8    </head>
9    <body>
10     <div id="container">
11
```

```
      <button type="button" @click="test1">测试1</button>
    </div>
    <script type="text/javascript">
      var vm = new Vue({
        el:"#container",
        methods:{
          test1:function(){
            //发送异步请求
            //  axios.get(url).then(fn);
            //  axios.get(url,{}).then(fn)
            axios.get("http://localhost:9999/music/detail",{
              params:{
                id:"25640392"
              }
            })
            .then(function(res){
              console.log(res);
            });
          }
        }
      });
    </script>

  </body>
</html>
```

12.3 axios异步请求方法

> axios提供了多种异步请求方法，实现对RESTful风格的支持

12.3.1 get请求

- axios.get(url).then(fn);
- axios.get(url,{}).then(fn)

```
//使用axios的get请求传递参数，需要将参数设置在params下
axios.get("http://localhost:9999/music/detail",{
    params:{
        id:"25640392"
    }
})
    .then(function(res){
    console.log(res);
});
```

12.3.2 post请求

- axios.post(url,{}).then(fn)

```
axios.post("http://localhost:9999/music/search",{s:"阿刁"})
    .then(function(res){
    console.log(res);
});
```

12.3.3 自定义请求

> 自定义请求：自定义请求方式、请求参数、请求头、请求体（post）

```
axios({
    url:"http://localhost:9999/music/search",
    method:"post",
    params:{
        //设置请求行传值
        s:"成都",
        limit:15
    },
    headers:{
        //设置请求头
    },
    data:{
        //设置请求体（post/put）
    }
}).then(function(res){
    console.log(res)
});
```

12.3.4 其他

- delete
- put
- option

12.4 并发请求

```
<div id="container">
    <button type="button" @click="test1">测试1</button>
</div>
<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        methods:{
            test1:function(){
                //发送异步请求

axios.all([listMusics(),getMusicDetail()]).then(axios.spread(function (r1, r2) {
                    // 两个请求现在都执行完成
                    console.log(r1);
                    console.log(r2);
                }));
            }
        }
    });

    function listMusics() {
        return axios.get('http://localhost:9999/music/search?s=成都');
    }

    function getMusicDetail() {
        return axios.get('http://localhost:9999/music/detail?id=25640392');
    }
</script>
```

12.5 箭头函数

12.5.1 axios回调函数的参数res

> res并不是接口返回的数据，而是表示一个响应对象；res.data才表示接口响应的数据

12.5.2 箭头函数

```
<script type="text/javascript">
    var vm = new Vue({
        el:"#container",
        data:{
            song:{

            }
        },
        methods:{
            test1:function(){

                //发送异步请求
                axios.get("http://localhost:9999/music/detail?id=25640392").then( (res)=>{
                    // res并不是接口返回的数据，而是表示一个响应对象；res.data才表示接口响应的数据
                    if(res.data.code == 200){
                        this.song = res.data.songs[0];
                    }
                });

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

十三、路由 router

> router是由vue官方提供的用于实现组件跳转的插件

13.1 路由插件的引用

13.3.1 离线

```
<script type="text/javascript" src="js/vue.js" ></script>
<script type="text/javascript" src="js/vue-router.js"></script>
```

13.3.2 在线CDN

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

13.2 路由使用案例

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
    <style type="text/css">
      body{padding: 0px;margin: 0px;}
      ul{list-style: none;}
      ul li{display: inline; float: left; margin-left: 15px;  margin-bottom: 15px;}
      ul li a{text-decoration: none; color: white; font-size: 18px; font-weight: bold;}
      ul li a:hover{color: yellow;}
    </style>
    <script type="text/javascript" src="js/vue.js" ></script>
    <script type="text/javascript" src="js/vue-router.js"></script>
  </head>
  <body>

    <div id="container">
      <div style="width: 100%; height: 70px; background: #00BFFF;">
        <table>
          <tr>
          <td><img src="img/logo.png" height="70" style="margin-left:100px;"/></td>
            <td>
              <ul>
                <li><router-link to="/a">首页</router-link></li>
```

```
                <li><router-link to="/b">Java</router-link></li>
                <li><router-link to="/c">HTML5</router-link></li>
                <li><router-link to="/d">Python</router-link></li>
              </ul>
            </td>
          </tr>
        </table>
      </div>
      <div style="width: 100%; height: 680px; background:
lemonchiffon;">
        <router-view></router-view>
      </div>
    </div>
    <script type="text/javascript">
      // vue的路由旨在为单页面应用开发提供便捷
      //1.定义链接跳转的模板（组件）
      const t1 = {template:`<p>index</p>`};
      const t2 = {template:`<p>Java</p>`};
      const t3 = {template:`<p>HTML5</p>`};
      const t4 = {template:`<p>PYTHON</p>`};

      const myrouter = new VueRouter({
        routes:[
          {path:"/a",component:t1},
          {path:"/b",component:t2},
          {path:"/c",component:t3},
          {path:"/d",component:t4}
        ]
      });

      var vm = new Vue({
        el:"#container",
        router:myrouter
      });
    </script>

  </body>
</html>
```

13.3 动态路由匹配

13.3.1 通配符

> `*`可以匹配任意路径

例如:

- `/user-*` 匹配所有以`user-`开头的任意路径
- `/*` 匹配所有路径

```
const myrouter = new VueRouter({
    routes:[
        {path:"/user-*",component:...},
        {path:"/*",component:...}
    ]
});
```

`注意`如果使用通配符定义路径，需要注意路由声明的顺序

13.3.2 路由参数

- `/a/:id` 可以匹配 `/a/`开头的路径

```
<div id="container">
    <li><router-link to="/a/101">首页</router-link></li>
    <router-view></router-view>
</div>

<script type="text/javascript">
    const t1 = {template:`<p>index:{{$route.params.id}}</p>`};

    const myrouter = new VueRouter({
        routes:[
            {path:"/a/:id",component:t1}
        ]
    });

    var vm = new Vue({
        el:"#container",
        router:myrouter
    });
</script>
```

13.3.3 优先级

如果一个路径匹配了多个路由，则按照路由的配置顺序：路由定义的越早优先级就越高。

13.4 嵌套路由

> 在一级路由的组件中显示二级路由

```
<div id="container">
    <router-link to="/a">首页</router-link>
    <router-link to="/a/c1">首页-c1</router-link>
    <router-link to="/a/c2">首页-c2</router-link>
    <router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>index<hr/><router-view></router-view></div>"
    };

    const t2 = {template:`<div>t2</div>`};
    const t3 = {template:`<div>t3</div>`};


    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                component:t1,
                children:[
                    {
                        path:"c1",
                        component:t2
                    },
                    {
                        path:"c2",
                        component:t3
                    }
                ]
            }
        ]
    });

```

```
    var vm = new Vue({
        el:"#container",
        router:myrouter
    });
</script>
```

13.5 编程式导航

13.5.1 push()

```
<div id="container">
    <button type="button" @click="test">按钮</button>
    <router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>index</div>"
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                component:t1
            }
        ]
    });

    var vm = new Vue({
        el:"#container",
        router:myrouter,
        methods:{
            test:function(){
                //js代码实现路由跳转：编程式导航
                myrouter.push("/a");
            }
        }
    });
</script>
```

13.5.2 push()参数

```
//1.字符串
myrouter.push("/a");

//2.对象
myrouter.push({path:"/a"});

//3.命名的路由  name参数指的是定义路由时指定的名字
myrouter.push({name:"r1",params:{id:101}});

//4.URL传值，相当于/a?id=101
myrouter.push({path:"/a",query:{id:101}});
```

13.5.3 replace()

> 功能与push一致，区别在于replace()不会向history添加新的浏览记录

13.5.4 go()

> 参数为一个整数，表示在浏览器历史记录中前后/后退多少步 相当于`window.history.go(-1)`的作用

13.6 命名路由

> 命名路由：在定义路由的时候可以给路由指定name，我们在进行路由导航时可以通过路由的名字导航

```
<div id="container">
    <input type="text" v-model="rname"/>
    <router-link :to="{name:rname}">t1</router-link>
    <button type="button" @click="test">按钮1</button>
    <router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>t1</div>"
    };

    const t2 = {
        template:"<div style='width:400px; height:200px; border:red 1px solid'>t2</div>"
```

```
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                name:"r1",
                component:t1
            },
            {
                path:"/b",
                name:"r2",
                component:t2
            }
        ]
    });

    var vm = new Vue({
        el:"#container",
        data:{
            rname:"r1"
        },
        router:myrouter,
        methods:{
            test:function(){
                myrouter.push({name:vm.rname});
            }
        }
    });
</script>
```

13.7 命名路由视图

```
<div id="container">
    <router-link to="/a">t1</router-link>
    <router-link to="/b">t2</router-link>

    <!--路由视图-->
    <!--如果在HTML中有一个以上的路由视图router-view，需要给router-view指定name，在路由中使用components映射多个组件根据name设置组件与router-view绑定关系-->
    <router-view name="v1"></router-view>
```

```
    <router-view name="v2"></router-view>
</div>
<script type="text/javascript">
    const t11 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>t11</div>"
    };
    const t12 = {
        template:"<div style='width:400px; height:200px; background:pink'>t12</div>"
    };

    const t21 = {
        template:"<div style='width:400px; height:200px; border:red 1px solid'>t21</div>"
    };
    const t22 = {
        template:"<div style='width:400px; height:200px; background:yellow'>t22</div>"
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                components:{
                    v1:t11,
                    v2:t12
                }
            },
            {
                path:"/b",
                components:{
                    v1:t21,
                    v2:t22
                }
            }
        ]
    });

    var vm = new Vue({
        el:"#container",
```

```
        router:myrouter
    });
</script>
```

13.8 重定向和别名

13.8.1 重定向

访问`/b`，重定向到`/a`

```
<div id="container">
    <router-link to="/a">路径A</router-link>
    <router-link to="/b">路径B</router-link>
    <router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>index</div>"
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                component:t1
            },
            {
                path:"/b",
                redirect:"/a"
            }
        ]
    });

    var vm = new Vue({
        el:"#container",
        router:myrouter
    });
</script>
```

- 根据路由命名重定向

```
const myrouter = new VueRouter({
    routes:[
        {
            path:"/a",
            name:"r1",
            component:t1
        },
        {
            path:"/b",
            //redirect:"/a"    //根据路由路径重定向
            redirect:{name:"r1"}  //根据路由命名重定向
        }
    ]
});
```

13.8.2 路由别名

```
<div id="container">
    <router-link to="/a">路径A</router-link>
    <router-link to="/wahaha">路径wahaha（别名）</router-link>
    <router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:"<div style='width:400px; height:200px; border:blue 1px solid'>index</div>"
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a",
                alias:"/wahaha",
                component:t1
            }
        ]
    });

    var vm = new Vue({
        el:"#container",
```

```
        router:myrouter
    });
</script>
```

13.9 路由组件传参

可以通过`/url/:attr`方式实现通过路由传值给组件

```
<div id="container">
    <router-link to="/a/101">路径A</router-link>
<router-view></router-view>
</div>
<script type="text/javascript">
    const t1 = {
        template:`<div style='width:400px; height:200px; border:blue 1px solid'>
                    index: {{$route.params.id}}
                    </div>`
          };

        const myrouter = new VueRouter({
            routes:[
                {
                    path:"/a/:id",
                    component:t1
                }
            ]
        });

        var vm = new Vue({
            el:"#container",
            router:myrouter
        });
</script>
```

通过props传参

```
<div id="container">
    <router-link to="/a/102">路径A</router-link>
    <router-view></router-view>
</div>
<script type="text/javascript">
```

```
    const t1 = {
        props:["id"],
        template:`<div style='width:400px; height:200px; border:blue 1px solid'>
        index: {{id}}
          </div>`
    };

    const myrouter = new VueRouter({
        routes:[
            {
                path:"/a/:id",
                props:true,
                component:t1

            }
        ]
    });

    var vm = new Vue({
        el:"#container",
        router:myrouter
    });
</script>
```

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静!