1.理解MVVM思想

什么是MVVM？MVVM是Model-View-ViewModel的缩写。

MVVM最早由微软提出来，它借鉴了桌面应用程序的MVC思想，在前端页面中，把Model用纯JavaScript对象表示，View负责显示，两者做到了最大限度的分离。

把Model和View关联起来的就是ViewModel。ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model。

2.vue.js介绍

- Vue是一套用于构建用户界面的渐进式的轻量级框架
- Vue是MVVM设计模式的具体实现方案
- Vue只关注视图层，便于与第三方库或既有项目整合
- Vue可以与现代化的工具链和各种类库结合使用，为复杂的SPA提供驱动

3.vue起步

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    {{message}} <!-- 文本差值表达式 模板语法 {{数据变量名称}}-->
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            const message = ref("hello vue");
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                message
            }
        }
    });
```

```
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

4.模板语法

- Vue.js中使用的是基于html的模版语法，可以将Dom元素绑定至Vue实例中的数据。
- 页面中模版即就是通过{{}}方式来实现模版数据绑定。

4.1 数据绑定的方式

- 单向绑定
    - 方式1：使用两对大括号{{}}
    - 方式2：使用v-text、v-html
    - 方式3：v-once 数据只绑定一次

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <!-- 模板语法 -->
    <!-- 1、单向绑定 双大括号语法 插入文本值 -->
    <p>单向绑定大号括号语法：{{message}}</p>
    <!-- v-html指令 插入html代码 -->
    <p>使用v-html指令 插入HTML代码: <span v-html="rawHtml"></span></p>
    <!-- v-text指令 插入文本 -->
    <p>v-text指令 插入文本: <span v-text="rawHtml"></span></p>
    <!-- v-once指令 表示数据只绑定一次 -->
    <p v-once>v-once指令:{{message}} </p>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数
对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发
生时响应式地更新 DOM。
            const message = ref("hello vue");
            const rawHtml = ref("<a href='http://www.baidu.com'>百度</a>");
```

```
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                message,
                rawHtml
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

- 双向绑定：
  - v-model指令实现

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <!-- 单行文本 -->
    <input type="text" v-model="message"/>
    <p>message:{{message}}</p>
    <!-- 多行文本 -->
    <textarea v-model="message"></textarea>
    <p>message:{{message}}</p>
    <!-- 单选框 -->
    <input type="radio" v-model="gender" value="男"/>男
    <input type="radio" v-model="gender" value="女"/>女
    <p>gender:{{gender}}</p>
    <!-- 复选框 -->
    <input type="checkbox" v-model="hobbies" value="吃"/>吃
    <input type="checkbox" v-model="hobbies" value="喝"/>喝
    <input type="checkbox" v-model="hobbies" value="玩"/>玩
    <input type="checkbox" v-model="hobbies" value="乐"/>乐
    <p>hobbies:{{hobbies}}</p>
    <!-- 下拉菜单 -->
    <select v-model="grade">
        <option value="本科">本科</option>
        <option value="专科">专科</option>
        <option value="高中">高中</option>
    </select>
    <p>grade:{{grade}}</p>
    <hr/>
    <!-- 在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步 -->
    <!-- 你可以添加 lazy 修饰符，从而转为在 change 事件_之后_进行同步 -->
    <input type="text" v-model.lazy="message"/>
    <p>message:{{message}}</p>
```

```
    <!-- 可以自动去除用户输入的字符首尾的空白字符，可以给v-model添加trim修饰符：-->
    <input type="text" v-model.trim="message"/>
    <p>message:{{message}}</p>
    <!-- 转换用户输入的值为数值类型 ，可以给v-model添加number修饰符：-->
    <input type="text" v-model.number="message"/>
    <p>message:{{message}}</p>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            let message = ref("初始值");
            let gender = ref("男");
            let hobbies = ref(["吃"]);
            let grade = ref("本科");
            let age = ref(15);
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                message,
                gender,
                hobbies,
                grade,
                age
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

5.常用指令

5.1 v-bind

- 1、v-bind
  - 使用v-bind指令进行DOM属性绑定
- v-bind 用法：
  - v-bind:属性名="属性值"
  - 简写　:属性名="属性值"
    - :src=""
- style和class属性用法：
  - style:

        - .txt{color:#ff0000;} .bgcolor{background-color: #ccc;}
    - html:
        - `<span :class="{txt:true,bgcolor:true}">文本内容</span>`

- v-bind属性数据绑定代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <p v-bind:id="dynamicId">使用v-bind指令实现属性数据绑定</p>
    <p :id="dynamicId">使用v-bind指令的简写方式实现属性数据绑定</p>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            const dynamicId = ref("E201503");
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                dynamicId
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

v-bind属性操作class和style的实例代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style type="text/css">
        .active{
            color: #ff0000;
        }
```

```
        .bgColor{
            background-color: #ccc;
        }
    </style>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <div :class="{active:isActive,bgColor:isBgColor}">绑定class</div>
    <div :class="[isActive?'active':'',isBgColor?'bgColor':'']">数组绑定
class</div>
    <div :style="{color:activeColor,fontSize:fontSize}">绑定内联样式</div>
    <div v-bind:style="styleObject">绑定内联样式（绑定样式对象，代码更清晰）</div>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, reactive} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义基本和复杂类型数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响
应式地更新 DOM。
            const isActive = ref(true);
            const isBgColor = ref(true);
            const activeColor = ref('blue');
            const fontSize = ref('30px');
            //定义复杂数据（例如对象）,reactive() 函数来声明响应式状态
            const styleObject = reactive({
                fontSize: "30px",
                color: "green"
            });
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                isActive,
                isBgColor,
                activeColor,
                fontSize,
                styleObject
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

5.2 v-if、v-elseif、v-else

- v-if、v-elseif、v-else

- v-if 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 true值的时候被渲染。
- 可以使用v-else-if指令来充当 v-if 的“else-if 块”，可以连续使用。
- 可以使用 v-else 指令来表示 v-if 的“else块”。
- v-else 元素必须紧跟在带 v-if 或者 v-else-if 的元素的后面，否则它将不会被识别。
- v-else-if 也必须紧跟在带 v-if 或者 v-else-if 的元素之后。
- 控制切换一个元素是否显示也相当简单：

- 实例代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <div v-if="isBl">显示</div>
    <div>哈哈</div>
    <!-- v-else必须紧跟在v-if或者v-else-if后面-，否则不显示-->
    <div v-else>不显示</div>

    <div v-if="type==='A'">A</div>
    <div v-else-if="type==='B'">B</div>
    <div v-else-if="type==='C'">C</div>
    <div v-else>非ABC</div>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            const isBl = ref(false);
            const type = ref('B');
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                isBl,
                type
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
```

```
</html>
```

5.3 v-show

- v-show
  - v-show 按条件显示一个元素的指令
  - `v-show` 会在 DOM 渲染中保留该元素；`v-show` 仅切换了该元素上名为 `display` 的 CSS 属性。
  - `v-show` 不支持在 `<template>` 元素上使用，也不能和 `v-else` 搭配使用
- v-if与v-show
  - `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，如果需要频繁切换，则使用 `v-show` 较好；如果在运行时绑定条件很少改变，则 `v-if` 会更合适。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <div v-show="ok">你好</div>
    <div>大家好</div>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            //const ok = ref(true);
            const ok = ref(false);
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                ok
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

5.4 v-for

- v-for
  - 基于源数据多次渲染元素或模板块，对数组或对象进行循环操作。
  - 遍历一个数组和数组对象
- 实例代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <!-- 1、v-for遍历一个数组 -->
    <!-- v-for 指令需要使用item in items 形式的特殊语法，其中items是元数据数组
    而item 是正在被迭代的数组元素别名-->
    <ul>
        <li v-for="item in arr">
            {{item}}
        </li>
    </ul>
    <!-- 2、v-for支持第二个参数 index表示索引 -->
    <ul>
        <li v-for="(item,index) in arr">
            索引为：{{index}}，元素值为：{{item}}
        </li>
    </ul>
    <!-- 3、v-for遍历一个对象数组 -->
    <ul>
        <li v-for="item in objs">
            {{item.name}}
        </li>
    </ul>
    <!-- 4、v-for遍历一个对象数组 -->
    <ul>
        <li v-for="(item,index) in objs">
            {{index}}--{{item.id}}--{{item.name}}
        </li>
    </ul>

    <table border="1px" cellspacing="0px">
        <tr>
            <td>编号</td>
            <td>用户名</td>
        </tr>
        <tr v-for="item in objs">
            <td>{{item.id}}</td>
            <td>{{item.name}}</td>
        </tr>
```

```
    </table>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            const arr = ref(['a','b','c']);
            const objs = ref([
                {id:1,name:"A"},
                {id:2,name:"B"},
                {id:3,name:"C"}
            ]);
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                arr,
                objs
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

5.5 v-on

- v-on
    - 用来绑定事件，用法：v-on:事件="函数"，简写方式@事件="函数"

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<div id="app">
    <!-- v-on指令监听DOM事件 -->
    <input type="button" value="事件监听器1(v-on)" v-on:click="method1()"/>
    <!--模板中读取时不需要.value-->
    name:{{myName}}<br/>
    <input type="button" value="事件监听器2(v-on)" v-on:click="method2('参数')"/>
    username:{{obj.username}},password:{{obj.password}}<br/>
```

```
    <input type="button" value="事件监听器3(v-on)" v-on:click="method3('有返回值')"/>
    <input type="button" value="事件监听器4(v-on)" v-on:click="method4()"/>
    <ul>
        <li v-for="o in objs">{{o.username}}-{{o.password}}</li>
    </ul>
    <!-- v-on.once 表示只做一次-->
    <input type="button" value="只做一次事件绑定(v-on.once)" v-on:click.once="method4()"/>
    <!-- 使用指令v-on的缩写@绑定事件 属性指令v-bind缩写绑定属性 -->
    <input type="button" value="事件监听器5(v-on缩写@绑定)" @click="method1()"/>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, reactive} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            //定义数据,ref() 函数来声明响应式状态
            //响应式是Vue的核心功能之一，即自动跟踪 JavaScript 数据状态变化并在改变发生时响应式地更新 DOM。
            let myName = ref("KING");
            const obj = reactive({
                username: "jack",
                password: "12345"
            });
            //不是响应式，数据发生变化，页面不会更新
            const objs =reactive([
                {
                    username: "lucy",
                    password: "123"
                },
                {
                    username: "tom",
                    password: "321"
                }
            ]);

            const method1 = () => {
                console.log("method1无参函数");
                //ref响应式数据 获取和修改值需要使用value
                console.log(myName.value);
                myName.value = "张三";
            };
            const method2 = param => {
                console.log("method2有参函数，参数为param:" + param);
                //reactive响应式数据 获取和修改不需要value
                obj.username = "李四";
                obj.password = "88888";
            };
            const method3 = param => {
                console.log("method3有参数函数，参数param:" + param);
```

```
                return "返回值";
            };
            const method4 = () => {
                //调用当前实例的其他方法  this表示当前实例(method4所在的实例)
                var result = method3("method4调用method3");
                console.log("result:" + result);
                //调用当前的数据
                console.log(myName.value);
                console.log(obj);
                console.log(objs);

                objs[0].username="王五";
                objs[0].password='999999';
            };
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                myName,
                obj,
                objs,
                method1,
                method2,
                method3,
                method4
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

6.计算属性

- 计算属性
  - html元素中的表达式虽然方便，但也只能用来做简单的操作。如果在模板中写太多逻辑，会让模板变得臃肿，难以维护。所以推荐使用计算属性来描述依赖响应式状态的复杂逻辑。
  - 计算属性也是用来存储数据的，但具有以下几个特点：
    - 数据可以进行逻辑处理操作
    - 可以对计算属性中的数据进行监视
  - **注意：**
    - 计算属性是基于它的依赖进行更新的，只有在相关依赖发生改变时才能更新。
    - 计算属性是缓存的，只要相关依赖没有改变，多次访问计算属性得到的值是之前缓存的计算结果，不会多次执行
  - **get和set方法**
    - 计算属性由两部分组成：get和set，分别用来获取计算属性和设置计算属性的操作。
    - 默认只有get方法。
- 代码实例：

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

```
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<div id="app">
    <!-- 可以像普通属性一样在模板中绑定计算属性-->
    <p>{{fullName}}</p>
    <p>{{fullName}}</p>
    <!--调用多次函数，没有缓存，没有计算属性效率高-->
    <p>{{fullNameMethod()}}</p>
    <p>{{fullNameMethod()}}</p>
    <!-- 绑定2次数据，缓存关闭控制台打印2次，说明调用了2次 -->
    <p>{{myFullName}}</p>
    <p>{{myFullName}}</p>
    <input type="button" value="修改firstName值" @click="modifyFirstName">
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, computed} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            const firstName = ref('张');
            const lastName = ref('三');
            //计算属性
            //这个是计算属性get(计算属性只有一个函数就是默认get)
            const fullName = computed(() => {
                console.log("计算属性.....");
                return firstName.value + ' ' + lastName.value;
            });
            const fullNameMethod = () => {
                console.log("调用函数.....")
                return firstName.value + ' ' + lastName.value;
            };
            //计算属性getter和setter
            const myFullName = computed({
                // getter
                get() {
                    console.log("myFullName计算属性.....");
                    return firstName.value + ' ' + lastName.value
                },
                // setter
                set(newValue) {
                    // 注意：我们这里使用的是解构赋值语法
                    [firstName.value, lastName.value] = newValue.split(' ')
                }
            });
            const modifyFirstName = () => {
                //只给firstName进行赋值，所有的计算属性全部重新执行计算并更新显示值
                firstName.value = "李";
```

```
                //计算属性赋值
                //myFullName.value='赵 六';
            }
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                fullName,
                fullNameMethod,
                myFullName,
                modifyFirstName
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

7.侦听器

7.1 侦听器watch

侦听器watch在每次响应式状态发生变化时触发回调函数

watch的使用场景是：当在某个数据源发生变化时, 我们需要做一些操作, 或者当需要在数据变化时执行异步或开销较大的操作时. 我们就可以使用watch来进行监听。

watch的参数：

1、watch有3个参数：第一个参数为数据源，第二个参数为监听触发的回调函数，第三个参数是对象表示其他特性，如：是否深度监听、是否立即执行监听等

2、watch 的第一个参数可以是不同形式的“数据源”：它可以是一个 ref (包括计算属性)、一个响应式对象、一个 getter 函数、或多个数据源组成的数组：

watch分为普通侦听和深度侦听

普通侦听代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    x={{x}} <br/>
    doubleX={{doubleX}}<br/>
    y={{y}}<br/>
    <input type="button" @click="changeX" value="x自增"/>
    <input type="button" @click="changeY" value="y自增"/>
</div>
</body>
```

```
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, watch} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            let x = ref(1);
            let doubleX = ref(0);
            let y = ref(1);

            const changeX = () => {
                x.value++;
            }
            const changeY = () => {
                y.value++;
            }

            // 单个数据 ref，数据发生改变就会触发侦听回调函数
            //回调函数的第一个参数是新值 第二个参数是旧值
            watch(x, (newX,oldX) => {
                console.log(`oldX is ${oldX}`);
                console.log(`newX is ${newX}`);
                doubleX.value = 2 * x.value;
            }, {immediate: true})

            //侦听getter函数返回结果值，结果值发生改变就会触发侦听回调函数
            watch(() => x.value + y.value, (sum) => {
                    console.log(`sum of x + y is: ${sum}`)
                }
            )

            //多个来源组成的数组，数组中任何数据发生改变都会触发侦听回调函数
            watch([x, () => y.value], ([newX, newY]) => {
                console.log(`x is ${newX} and y is ${newY}`)
            })
            //通过return返回对象，把定义的数据或者函数暴露
            return {
                x,doubleX, y, changeX, changeY
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

注意，不能直接侦听响应式对象的属性值，例如:

```
const obj = reactive({ count: 0 })

// 错误，因为 watch() 得到的参数是一个 number
watch(obj.count, (count) => {
  console.log(`count is: ${count}`)
})
```

这里需要用一个返回该属性的 getter 函数：

```
const obj = reactive({ count: 0 });

// 提供一个 getter 函数
watch(
  () => obj.count,
  (count) => {
    console.log(`count is: ${count}`)
  }
)
```

7.2 理解immediate属性

如上watch有一个特点是: 第一次初始化页面的时候, 是不会去执行x这个属性监听的, 只有当x值发生改变的时候才会执行监听回调函数. 因此我们上面第一次初始化页面的时候, 'doubleX' 属性为0。那么我们现在想要第一次初始化页面的时候也希望它能够执行 'x' 进行监听, 最后能把结果返回给 'doubleX' 值来。因此我们需要添加下我们的 watch的函数的三个参数{immediate:true}, 代码如下所示:

```
.......
// 单个数据 ref，数据发生改变就会触发监听回调函数
// watch第三个参数 {immediate:true} 立即执行监听x
watch(x, (newX) => {
  console.log(`x is ${newX}`);
  doubleX.value = 2 * x.value;
},{immediate:true})
........
```

7.3 理解deep属性

watch里面第三个参数的属性deep为true，含义是：深度监听某个对象的属性的值。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    obj.count={{obj.count}}<br/>
    <input type="button" @click="changeCount" value="obj.count自增"/>
```

```
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, watch, reactive} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            let obj = reactive({count: 0});
            let obj2 = reactive({obj});


            const changeCount=()=>{
                obj.count++;
            }

            //侦听对象(深度侦听)，对象中的嵌套属性发生改变就会触发监听回调函数
            // watch(obj,(newValue,oldValue)=>{
            //     // 注意：`newValue` 此处和 `oldValue` 是相等的
            //     // 因为它们是同一个对象！
            //     console.log(newValue);
            //     console.log(oldValue);
            //     console.log(newValue===oldValue);
            // })

            //直接改变obj的count不会触发监听回调函数，因为只有obj地址改变 才会触发
            //可以使用第三个参数 {deep: true} 强制深度监听
            watch(()=>obj2.obj,(newValue,oldValue)=>{
                // 注意：`newValue` 此处和 `oldValue` 是相等的
                // 因为它们是同一个对象！
                console.log(newValue);
                console.log(oldValue);
                console.log(newValue===oldValue);
            },{deep:true})

            //通过return返回对象，把定义的数据或者函数暴露
            return {
                obj, changeCount
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

谨慎使用深度侦听：因为深度侦听需要遍历被侦听对象中的所有嵌套的属性，当用于大型数据结构时，开销很大。因此请只在必要时才使用它，并且要留意性能。

7.4 watch 和 computed的区别是：

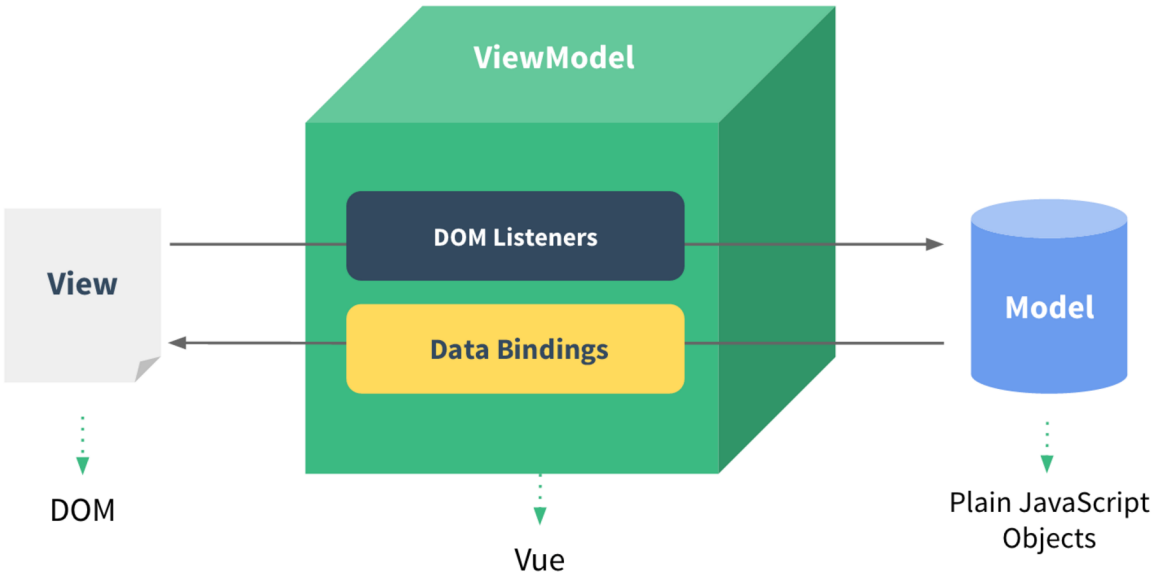
相同点：他们两者都是观察页面数据变化的。

不同点：computed只有当依赖的数据变化时才会计算，当数据没有变化时，它会读取缓存数据。watch每次都需要执行函数。watch更适用于数据变化时的异步操作。

8.VUE生命周期钩子

8.1 vue核心思想：数据驱动视图

一堆数据放在那里是不会有任何作用的，它必须通过我们的View Model（视图模型）才能操控视图。

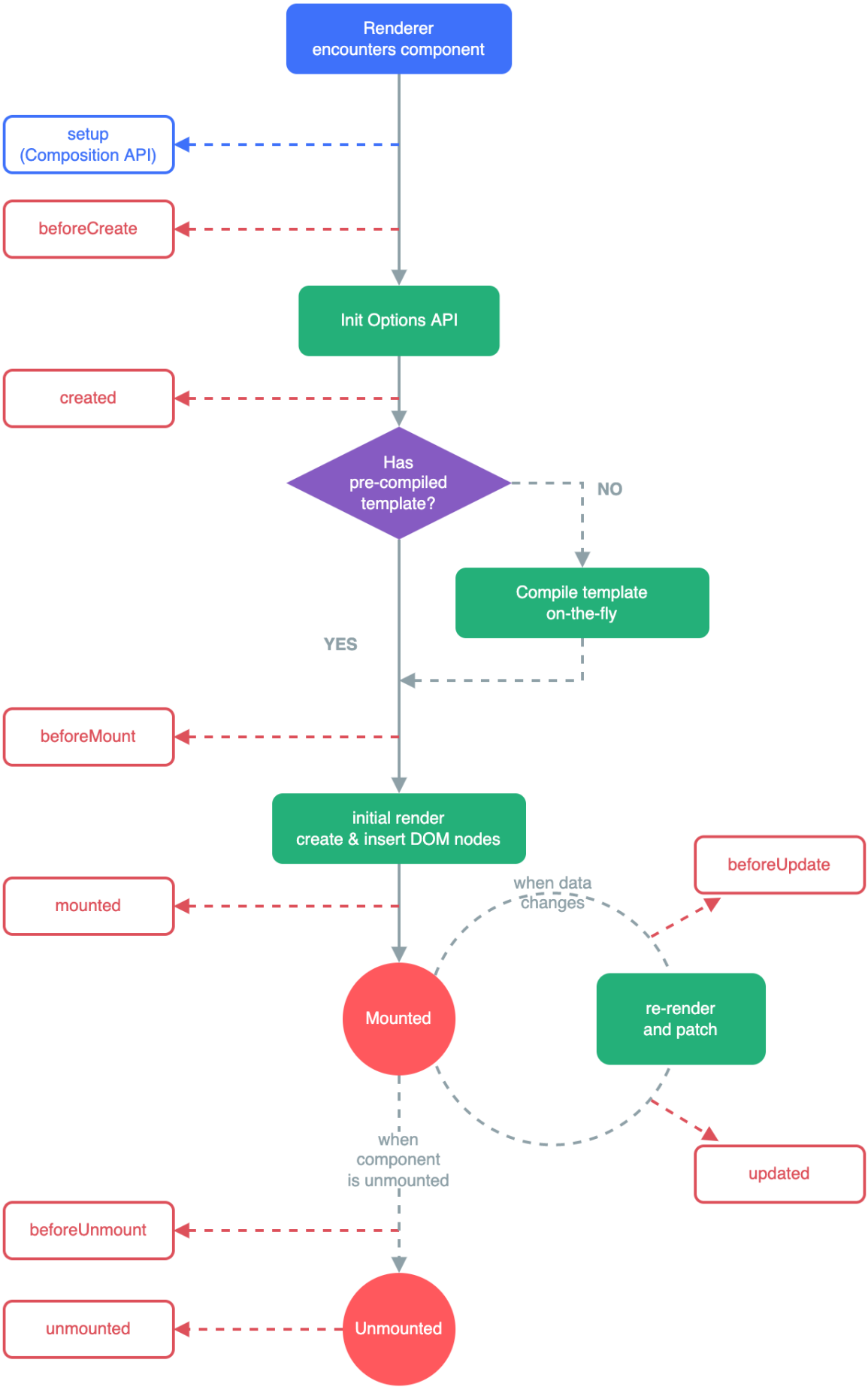


8.2 vue生命周期钩子

- 每个 Vue 组件实例在创建时都需要经历一系列的初始化步骤，比如设置好数据侦听，编译模板，挂载实例到 DOM，以及在数据改变时更新 DOM。在此过程中，它也会运行被称为生命周期钩子的函数，让开发者有机会在特定阶段运行自己的代码。
- 比如 created 钩子可以用来在一个实例被创建之后执行代码：

```
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp,onMounted} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是数据和函数对象
        setup() {
            onMounted(() => {
                console.log(`the component is now mounted.`)
            })
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
// => "the component is now mounted."
```

- vue实例从创建到销毁的过程，称为生命周期，共有九个阶段




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

```
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    message={{message}} <br/>
    <input type="button" @click="changeMsg" value="修改message"/>
    <input type="button" @click="unmountApp" value="卸载app"/>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref, onBeforeMount, onMounted, onBeforeUpdate, onUpdated,
onBeforeUnmount, onUnmounted} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup:开始创建组件之前，在 beforeCreate 和 created 之前执行，创建的是 data 和
method
        setup() {
            let message = ref("hello");

            const changeMsg = () => {
                message.value = "你好" + Date.now();
                console.log("改变message的值");
            }

            const unmountApp = () => {
                console.log(app);
                app.unmount();
            }


            //执行onXXX函数，把自定义回调函数绑定到Vue的声明周期的钩子函数上
            onBeforeMount(() => {
                console.log("组件挂载到节点上之前执行的函数======");
            })
            onMounted(() => {
                console.log("组件挂载到节点上之前执行的函数======");
            })
            onBeforeUpdate(() => {
                console.log("组件更新之前执行的函数======");
            })
            onUpdated(() => {
                console.log("组件更新完成之后执行的函数======");
            })
            onBeforeUnmount(() => {
                console.log("组件卸载之前执行的函数======");
            })
            onUnmounted(() => {
                console.log("组件卸载完成后执行的函数======");
            })
            return {
```

```
                message, changeMsg,unmountApp
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```

9.模版内元素的ref

- 虽然 Vue 的声明性渲染模型为你抽象了大部分对 DOM 的直接操作，但在某些情况下，我们仍然需要直接访问底层 DOM 元素。要实现这一点，我们可以使用特殊的 ref 属性
- ref 是一个特殊的属性。它允许我们在一个特定的 DOM 元素或子组件实例被挂载后，获得对它的直接引用。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- 引入vue的库文件 -->
    <script src="js/vue.js"></script>
</head>
<body>
<!-- 显示数据的层 -->
<div id="app">
    <input type="text" ref="myInput">
    <br/>
    <ul>
        <li v-for="item in list" ref="itemRefs">
            {{ item }}
        </li>
    </ul>
</div>
</body>
<script>
    //获取Vue对象的createApp和ref函数
    const {createApp, ref,onMounted} = Vue;
    //执行createApp函数返回app对象
    //createApp函数需要一个对象作为参数
    const app = createApp({
        //setup是vue的声明周期钩子函数， 执行时机：开始创建组件之前，创建的是 数据和函数对象
        setup() {
            // 声明一个 ref 来存放该元素的引用
            // 必须和模板里的 ref 同名
            let myInput = ref(null);
            let itemRefs = ref([]);
            //定义数据
            const list=ref([10,20,30]);
```

```
            //注意，你只可以在组件挂载后才能访问模板元素的引用。如果你想在模板中的表达式上访问input，在初次渲染时会是 null。
            // 这是因为在初次渲染前这个元素还不存在呢！
            onMounted(() => {
                console.log(myInput);
                //myInput.value 就是模版的 <input type="text" ref="myInput">元素的DOM对象
                myInput.value.value = "hahaha";
                console.log("==========================")
                //itemRefs.value 就是模版的li元素数组的DOM对象
                console.log(itemRefs.value);
                console.log(itemRefs.value[0]);
            });
            return {
                myInput,list,itemRefs
            }
        }
    });
    //绑定一个页面已经存在的DOM元素作为app对象挂载目标
    app.mount("#app");
</script>
</html>
```