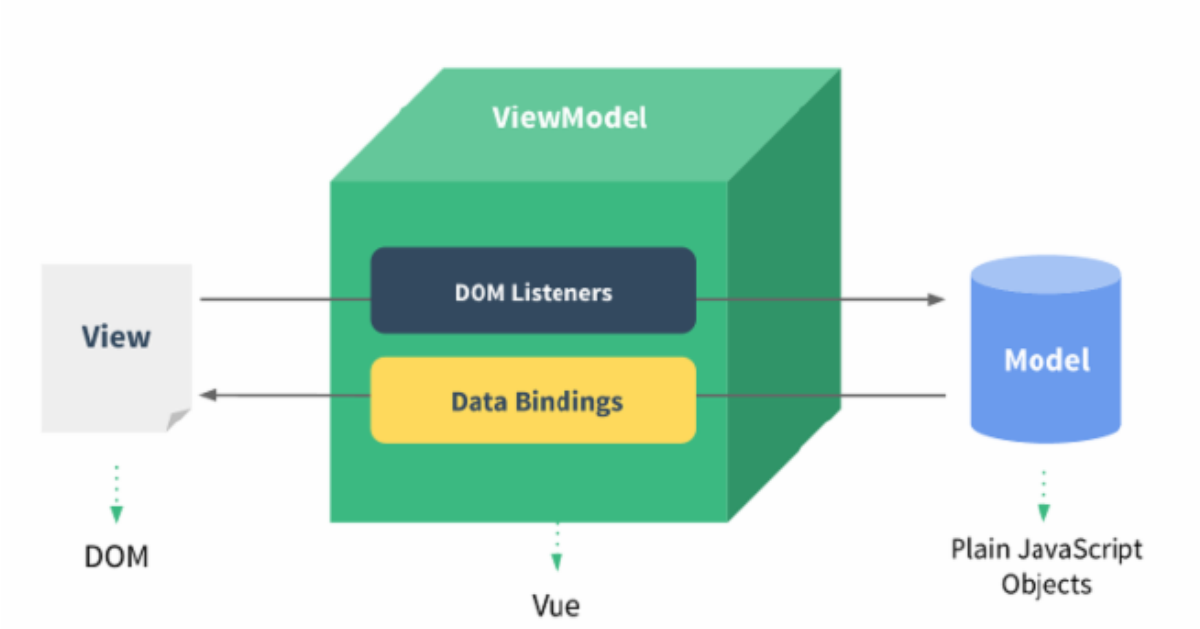
1. MVVM模型

MVVM是Model-View-ViewModel的简写,它本质上就是MVC的改进版,MVVM就是将其中的View的状态和行为抽象化,让我们将视图UI和业务逻辑分开.

MVVM模式和MVC模式一样,主要目的是分离视图(View)和模型(Model).

Vue.js是一个提供了MVVM风格的双向数据绑定的javascript库,专注于View层.它的核心是MVVM中的VM.也就是ViewModel,View负责连接View和Model,保证视图和数据的一致性.这种轻量级的框架让前端开发更加高效,便捷.



2. 插值表达式

```
<div id="div01">
    <div>{{message}}</div>
</div>
<script>
    new Vue({
        el:"#div01",//表示现在由vue来接管div01
        data:{
            message:"haha"
        }
    })
</script>
```

数据绑定最常见的形式就是使用"Mustache"语法(双大括号)的文本插值,Mustache标签将会被替代为对应数据对象上属性的字,无论何时,绑定的数据对象上属性发生了改变,擦值出的内容都会更新.

如:`{{number+1}}` `{{ok?'yes':'no'}}` 这些表达式会在所属Vue实例的数据作用域下作为JavaScipt被解析.有个限制就是,每个绑定都只能包含单个表达式.所以下面的例子都不会生效.

`{{var a = 1}}` 这是语句,不是表达式

`{{if(ok){return massage}}}` 流控制也不会生效,使用三元运算符

3.Vue的常用系统指令

3.1.v-on

所有的V-on都可以使用@进行简写:如v-on:click可以简写成@click.

1.v-on:click

```
<div id="div01">
    {{message}}
    <button v-on:click="fun('good')">点我试试</button>
</div>
<script>
    new Vue({
        el:"#div01",
        data:{
            message:"hello"
        },
        methods:{
            fun:function(data){
                this.message = data;
            }
        }
    });
</script>
```

2.v-on:keydown

```
<div id="div01">
    <input type="text" v-on:keydown = "fun01()"/>
</div>
<script>
    new Vue({
        el:"#div01",
        methods:{
            fun01:function(){
                var code = event.keyCode;
                if(!code>=48&&code<=57){
                    event.preventDefault();
                }
            }
        }
    })
</script>
```

如下是对preventDefault( )的解释.

语法

```
event.preventDefault()
```

说明

该方法将通知 Web 浏览器不要执行与事件关联的默认动作（如果存在这样的动作）。例如，如果 type 属性是 "submit"，在事件传播的任意阶段可以调用任意的事件句柄，通过调用该方法，可以阻止提交表单。注意，如果 Event 对象的 cancelable 属性是 fasle，那么就没有默认动作，或者不能阻止默认动作。无论哪种情况，调用该方法都没有作用。

3.v-on:mouseover

```
        <div id="div02">
            <div id="div01"  v-on:mouseover = "fun01()">
                <textarea id="textarea01" v-on:mouseover = "fun02()">这个地方是一个文件域
</textarea>
            </div>
        </div>
        <script>
            new Vue({
                el:"#div02",
                methods:{
                    fun01:function(){
                        alert("呵呵哒");

                    },
                    fun02:function(){
                        alert("你好呀");
                        event.stopPropagation();
                    }
                }
            });
        </script>
```

如下是对stopPropagation()方法的解释

不再派发事件。

终止事件在传播过程的捕获、目标处理或起泡阶段进一步传播。调用该方法后，该节点上处理该事件的处理程序将被调用，事件不再被分派到其他节点。

语法

```
event.stopPropagation()
```

4.事件修饰符

Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节，如：event.preventDefault() 或 event.stopPropagation()。

Vue.js通过由点(.)表示的指令后缀来调用修饰符。

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

5.按键修饰符

Vue 允许为 v-on 在监听键盘事件时添加按键修饰符

全部的按键别名：

- `.enter`
- `.tab`
- `.delete` (捕获 "删除" 和 "退格" 键)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`
- `.ctrl`
- `.alt`
- `.shift`
- `.meta`

3.2.v-text和v-html

```
<div id="div01">
    <div v-text="message"></div>
    <div v-html="message"></div>
</div>
<script>
    new Vue({
        el:"#div01",
        data:{
            message:"<h1>你好呀</h1>"
        }
    })
</script>
```

3.3.v-bind

```
<div id="div01">
    <font v-bind:color="hehe">黑马</font><br/>
    <font v-bind:size="didi">张三</font>
```

```
    <button v-on:click="fun01()">点我试试</button>
</div>
<script>
    new Vue({
        el:"#div01",
        data:{
            hehe:"red",
            didi:"10"
        },
        methods:{
            fun01:function(){
                this.hehe = "black";
                this.didi = "5";
            }
        }
    })
</script>
```

3.4.v-model

```
<div id="div01">
    <input type = "text" id = "username" v-model = "username"/>
    <input type = "text" id = "password" v-model = "password"/>
    <button id = "but01" v-on:click = "fun()">点我</button>
</div>
<script>
    new Vue({
        el:"#div01",
        data:{
            username : "",
            password : ""
        },
        methods : {
            fun : function(){
                alert("aaa");
                this.username = "张三";
                this.password = "lisi";
            }
        }
    });
</script>
```

3.5.v-for

```
<!-- 操作数组 -->
<div id="div01">
    <ul>
        <li v-for="(i,index) in list">{{index+"----"+i}}</li>
        <!-- 括号中谁在前面,谁就是value,谁在后面谁就是index -->
    </ul>
```

```
            <script>
                new Vue({
                    el : "#div01",
                    data : {
                        list : [12,13,14,15]
                    }
                })
            </script>
        </div>

        <!-- 操作对象 -->
        <div id = "div02">
            <ul>
                <li v-for = "(value,key) in product">{{key+"---"+value}}</li>
            </ul>
            <script>
                new Vue({
                    el : "#div02",
                    data : {
                        product : {
                            name : "jaja",
                            age : "didi"
                        }
                    }
                })
            </script>
        </div>

        <!-- 操作对象数组 -->
        <div id = "div03">
            <table border="1">
                <tr>
                    <th>编号</th>
                    <th>姓名</th>
                    <th>年龄</th>
                </tr>
                <tr v-for="p in products">
                    <td>{{p.id}}</td>
                    <td>{{p.name}}</td>
                    <td>{{p.age}}</td>
                </tr>
            </table>
        </div>
        <script>
            new Vue({
                el:"#div03",
                data:{
                    /* 需要注意的是:字符串一定要用引号引起来 */
                    products:[{id:1, name:"张三", age:3},{id:2, name:"赵六", age:3},{id:3,
name:"王五", age:3}]
                }
            })
        </script>
```

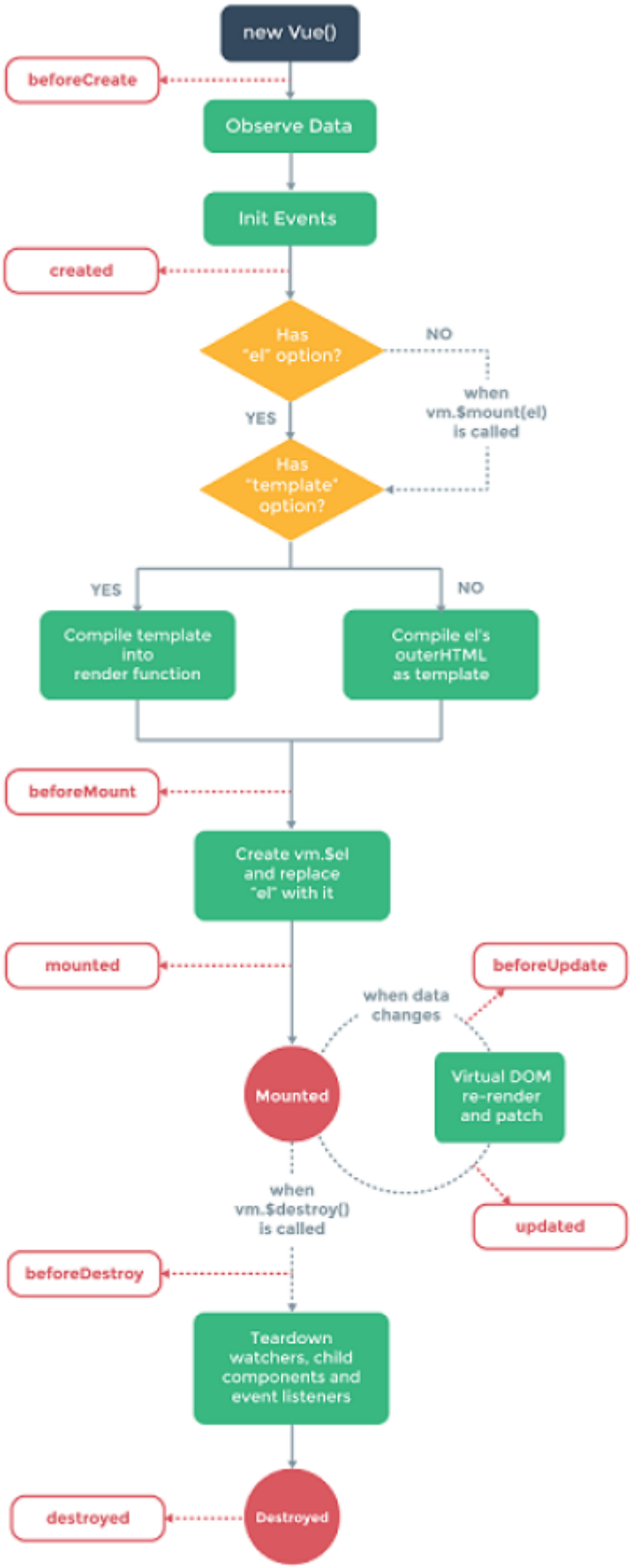
3.6.v-if与v-show

v-if是根据表达式的值来决定是否渲染元素

v-show是根据表达式的值来切换元素的display css属性

```
<div id="div01">
        <div v-if="flag">你好呀</div>
        <div v-show="flag">哈哈</div>
        <button v-on:click="fun01()">点我试试</button>
</div>
<script>
        new Vue({
                el : "#div01",
                data : {
                        flag :false
                },
                methods : {
                        fun01 : function( ){
                        this.flag = !this.flag;
                        }
                }
        })
</script>
```

4.Vue的生命周期



vue对象初始化过程中，会执行到beforeCreate,created,beforeMount,mounted 这几个钩子的内容:

- beforeCreate： 数据还没有监听，没有绑定到vue对象实例，同时也没有挂载对象.

- created： 数据已经绑定到了对象实例，但是还没有挂载对象.
- beforeMount: 模板已经编译好了，根据数据和模板已经生成了对应的元素对象，将数据对象关联到了对象的 el 属性，el属性是一个HTMLElement对象，也就是这个阶段，vue实例通过原生的createElement等方法来创 建这个html片段，准备注入到我们vue实例指明的el属性所对应的挂载点.
- mounted:将el的内容挂载到了el，相当于我们在jquery执行了(el).html(el),生成页面上真正的dom，上面我们就会发现dom的元素和我们el的元素是一致的。在此之后，我们能够用方法来获取到el元素下的dom对象，并进行各种操作.
- 当我们的data发生改变时，会调用beforeUpdate和updated方法
    - beforeUpdate： 数据更新到dom之前，我们可以看到$el对象已经修改，但是我们页面上dom的数据还没有发生改变.
    - updated: dom结构会通过虚拟dom的原则，找到需要更新页面dom结构的最小路径，将改变更新到 dom 上面，完成更新.
- beforeDestroy,destroed :实例的销毁，vue实例还是存在的，只是解绑了事件的监听还有watcher对象数据 与view的绑定，即数据驱动.

5.Axios

Axios的GitHub地址该网站有很多的对于Axios的实例

get请求

```
// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .finally(function () {
    // always executed
  });

// Optionally the request above could also be done as
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .then(function () {
    // always executed
  });

// Want to use async/await? Add the `async` keyword to your outer function/method.
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

POST请求

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

6.Vue整合项目

Vue+Axios.js

```
new Vue({
    el: "#abc",
    data: {
        userList: [],
        user: {}
    },
    methods: {
        //查找所有
        findAll: function () {
            var _this = this;
            var url = "/user/findAll";
            axios.get(url).then(function (response) {
                console.log(response);
                _this.userList = response.data;
            }).catch(function (error) {
                console.log(error);
            })
        },
        //修改的数据回显
        findById: function (id) {
            var _this = this;
            var url = "/user/findById/" + id;
            axios.get(url).then(function (response) {
                console.log(response);
                _this.user = response.data;
                $("#myModal").modal("show");
            }).catch(function (error) {
                console.log(error);
            })
        },
        //修改数据
        update: function () {
            var _this = this;
            var url = "/user/update";
            axios.post(url, _this.user).then(function () {
                _this.findAll();
```

```
                $("#myModal").modal("hide");
            }).catch(function (error) {
                console.log(error);
            })
        }
    },
    created: function () {
        this.findAll();
    }
});
```

controller层

```
@RestController
@RequestMapping("/user")
public class UserController  {
    @Autowired
    private UserService userService;

    @RequestMapping("/findAll")
    public List<User> findAll(){
        return userService.findAll();
    }

    @RequestMapping("/findById/{id}")
    public User findById(@PathVariable("id") Integer id){
        return userService.findById(id);
    }

    @RequestMapping("/update")
    public void update(@RequestBody User user){
        userService.update(user);
    }
}
```

servcie层

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;

    @Override
    public List<User> findAll() {
        return userDao.findAll();
    }

    @Override
    public User findById(Integer id) {
        Optional<User> optionalUser = userDao.findById(id);
        User user = optionalUser.get();
        return user;
```

```
    }

    @Override
    public void update(User user) {
        userDao.save(user);
    }
}
```

dao层

```
public interface UserDao extends JpaRepository<User,Integer> {

}
```

domain

```
@Entity//自动建立映射关系
@Data//自动生成getter/setter方法
public class User {
    @Id
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private String sex;
    private String email;
}
```