# 目录

# Part I

# Vue2与Vue3

# Chapter 1

# Vue2基础

## 1.1 初识Vue

el用于指定当前Vue实例为哪个容器服务，值通常为CSS选择器字符串。
data中用于存储数据，数据供el所指定的容器去使用，可以写成函数，但是这里先写成一个对象。

```
<script type="text/javascript">
const x = new Vue({
el: '#root',
// el: document.getElementById('root')
data: {
name: 'Vue2',
}
})
</script>
```

- 想让Vue工作，就必须创建一个Vue实例，且要传入一个配置对象；
- root容器里的代码依然符合html规范，只不过混入了一些特殊的Vue语法；
- root容器里的代码被称为Vue模板；
- Vue实例和容器是一一对应的；
- 真实开发中只有一个Vue实例，并且会配合着组件一起使用；
- {{xyz}}中的xyz要写js表达式，且xyz可以自动读取到data中的所有属性；
- 一旦data中的数据发生改变，那么页面中用到该数据的地方也会自动更新；

## 1.2 插值语法

Vue模板语法有2大类：

1. 插值语法：
   - 功能：用于解析**标签体**内容；
   - 写法：{{xyz}}，xyz是js表达式，且可以直接读取到data中的所有属性。
2. 指令语法：
   - 功能：用于解析标签（包括：标签属性、标签体内容、绑定事件等）；
   - 举例：`v-bind:href="xyz"` 或简写为 `:href="xyz"`，zyx同样要写js表达式，且可以直接读取到data中的所有属性；（不能有空格）
   - 备注：Vue中有很多的指令，且形式都是：`v-xyz`，此处我们只是拿v-bind举个例子;

```
<body>
<div id="root">
<h1>Hello, 插值语法</h1>
<h2>Hello, {{name}}</h2>
<hr>
<h1>指令语法</h1>
<a v-bind:href="url">点击去连接</a>
<script type="text/javascript">
new Vue({
el: '#root',
data: {
name: 'Jack',
url: 'https://www.apple.com.cn'
}
})
</script>
</div>
</body>
```

## 1.3 数据绑定

Vue2中有2种数据绑定的方式：

1. 单向绑定(v-bind)：数据只能从data流向页面；

2. 双向绑定(v-model)：数据不仅能从data流向页面，还可以从页面流向data。

   注意点:

   - 双向绑定一般都应用在表单类元素上（如：input、select等）；
   - v-model:value 可以简写为 v-model，因为v-model默认收集的就是value值。

```
<body>
<div id="root">
v-bind: <input type="text" v-bind:value="name"><br>
v-model: <input type="text" v-model:value="name">
<br>
<!-- simplified -->
v-bind: <input type="text" :value="name"><br>
v-model: <input type="text" v-model="name">
</div>
<script type="text/javascript">
new Vue({
el: '#root',
data: {
name: 'Vue2',
}
})
</script>
</body>
```

## 1.4 el和data的两个中写法

1. el有2种写法

   - new Vue时候配置el属性;
   - 先创建Vue实例，随后再通过`vm.$mount('#root')`指定el的值。

2. data有2种写法

   - 对象式;
   - 函数式;

3. 如何选择：目前哪种写法都可以，以后学习到组件时，data必须使用函数式，否则会报错。

4. 一个重要的原则：由Vue管理的函数，一定不要写箭头函数，一旦写了箭头函数，this就不再是Vue实例了。

```
<body>
<div id="root">
<h1>Hello, {{name}}</h1>
</div>
<script type="text/javascript">
const v = new Vue({
// element way1 object
// el: '#root',
// data: way1
// data: {
//     name: 'Vue2',
// }
// data: way2 function
data: function () {
        // this is the instance object
        console.log('@@@', this)
        return {
                name: 'Vue2'
            }
    }
})
// element way2
console.log(v);
v.$mount('#root');
</script>
</body>
```

## 1.5 MVVM模型

1. M：模型(Model)：data中的数据；
2. V：视图(View)：模板代码；
3. VM：视图模型(ViewModel)：Vue实例；

data中所有的属性，最后都出现在了vm（实例对象）身上，vm身上所有的属性以及Vue原型上所有属性，在Vue模板中都可以直接使用。

```
<body>
<div id="root">
<h1>Hello, {{name}}</h1>
<p>{{_c}}</p>
    <p>{{$emit}}</p>
</div>
<script type="text/javascript">
Vue.config.productionTip = false

const v = new Vue({
        data: function () {
                return {
                        name: 'Vue2',
                        address: 'uus'
                    }
            }
    }).$mount('#root');
</script>
</body>
```

## 1.6 数据代理

### 1.6.1 object.defineProperty

```
<script type="text/javascript">
let age = 18;
let person = {
name: 'John',
sex: 'male'
}
Object.defineProperty(person, 'age', {
        // value: 18,
        // enumerable: true,//can be enumerated or not
        // writable: true,//can be modified or not
        // configurable: true,//can be deleted or not
        get() {
                console.log('read age');
```

```
                return age
            },
        set(value) {
                console.log('modify age');
                age = value;
            }
    })
// not enumerable
console.log(Object.keys(person))
console.log(person)
</script>
```

### 1.6.2 数据代理

数据代理：通过一个对象代理另一个对象中的属性的操作（读/写）。

```
let obj = {
x: 100
}
let obj2 = {
y: 200
}
Object.defineProperty(obj2, 'x', {
        get() {
                return obj.x
            },
        set(value) {
                bj.x = value
            }
    })
```

## 1.7 事件处理

### 1.7.1 事件的基本使用

- 使用v-on:xyz或@xyz绑定事件，其中xyz是事件名；
- 事件的回调需要配置在methods中，最终会在vm上；
- methods中配置的函数，不要使用箭头函数，否则this就是指向vm对象或者组件实例对象；

- `@click="demo"`和`@click="demo($click)"`效果一样，但是后者可以传参；