REliable, INtelligent & Scalable Systems

互联网应用开发技术

Web Application Development

第5课 WEB前端-VUE简介

Episode Five

Vue Tutorials

陈昊鹏

chen-hp@sjtu.edu.cn

Web Application Development

Vue

- From
  - Vue Guide
  - https://vuejs.org/v2/guide/
  - Vue教程
  - https://cn.vuejs.org/v2/guide/
- References
  - 基于Idea从零搭建一个最简单的vue项目
  - https://www.jianshu.com/p/9c1d4f8ed068
  - electron-netease-cloud-music
  - https://github.com/Rocket1184/electron-netease-cloud-music
  - Vue.js Examples
  - https://vuejsexamples.com

Introduction

- What is Vue.js?
  - Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces.

- Declarative Rendering

```
<div id="app">
 {{ message }}
</div>

Vue.createApp({
    data() {
        return {
            message: 'Hello Vue!'
        }
    }
}).mount('#app')
```

Demo 1

Introduction

- Declarative Rendering v-bind

```
<div id="app-2">
  <span v-bind:title="message">
     Hover your mouse over me for a few seconds
     to see my dynamically bound title!
  </span>
</div>

Vue.createApp({
    data() {
        return {
            message: 'You loaded this page on ' + new Date().toLocaleString()
        }
    }
}).mount('#app-2')
```

Demo 2

Introduction

REliable, INtelligent & Scalable Systems

- Conditionals v-if

```
<div id="app-3">
  <span v-if="seen">Now you see me</span>
</div>

Vue.createApp({
    data() {
        return {
            seen: false
        }
    }
}).mount('#app-3')
```

Demo 3

Introduction

REliable, INtelligent & Scalable Systems

- Loops v-for

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>

Vue.createApp({
    data() {
        return {
            todos: [
                { text: 'Learn JavaScript' },
                { text: 'Learn Vue' },
                { text: 'Build something awesome' }
            ]
        }
    },
    methods: {
        addItem() {
            this.todos.push({text: 'Learn React'})
        }
    }
}).mount('#app-4')
```

Demo 4

Introduction

- Handling User Input

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>

Vue.createApp({
    data() {
        return {
            message: 'Hello Vue.js!'
        }
    },
    methods: {
        reverseMessage() {
            this.message = this.message.split('').reverse().join('')
        }
    }
}).mount('#app-5')
```

Demo 5

Introduction

- Handling User Input

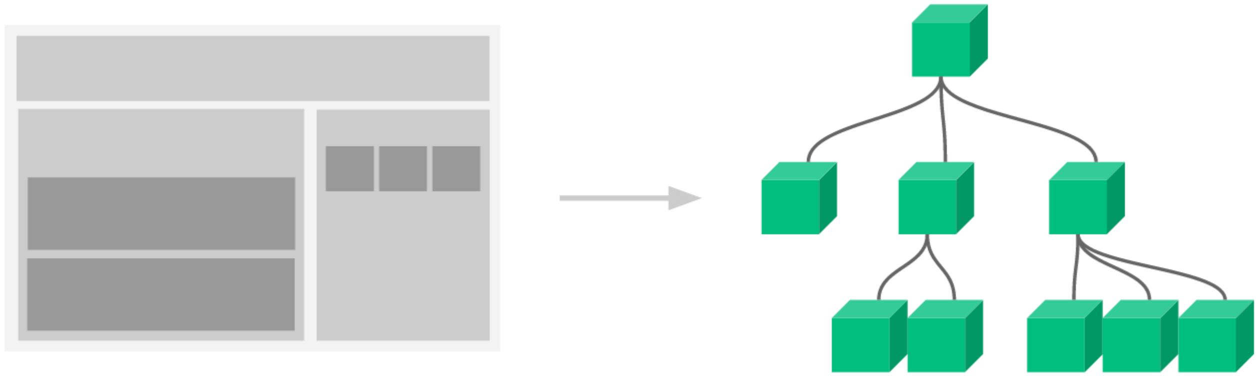
```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model:"message">
</div>

Vue.createApp({
    data() {
        return {
            message: 'Hello Vue!'
        }
    }
}).mount('#app-6')
```

Demo 6

Introduction

- Composing with Components



Introduction

- Composing with Components

```
<div id="app">
  <ol>
    <!--
       Now we provide each todo-item with the todo object
       it‘s representing, so that its content can be dynamic.
       We also need to provide each component with a “key”,
       which will be explained later.
    -->
    <todo-item
       v-for="item in groceryList"
       v-bind:todo="item"
       v-bind:key="item.id"
    ></todo-item>
  </ol>
</div>
```

Demo 7

Introduction

- Composing with Components

```
<script type="module">
    import TodoItem from './TodoItem.js'

    Vue.createApp({
        components: {
            TodoItem
        },
        data() {
            return {
                groceryList: [
                    { id: 0, text: 'Vegetables' },
                    { id: 1, text: 'Cheese' },
                    { id: 2, text: 'Whatever else humans are supposed to eat' }
                ]
            }
        }
    }).mount('#app')

</script>
```

Demo 7

Introduction

- Composing with Components
- TodoItem.js

```
export default {
    props: {
        todo: Object
    },
    template: '<li>{{ todo.text }}</li>'
}
```

Demo 7

The Vue Instance

- Create a Vue Instance

```
Vue.createApp({ // options })
```

- A Vue application consists of a root Vue instance created with Vue.createApp,
  - optionally organized into a tree of nested, reusable components.

```
Root Instance
 └─ TodoList
    ├─ TodoItem
    │  ├─ DeleteTodoButton
    │  └─ EditTodoButton
    └─ TodoListFooter
       ├─ ClearTodosButton
       └─ TodoListStatistics
```

The Vue Instance

- Data and Methods

```
Vue.createApp({
    data() {
        return {
            foo: 'bar'
        }
    }
}).mount('#app’)

<p>{{ foo }}</p>
<!-- this will no longer update `foo`! -->
<button v-on:click="foo = 'baz'">Change it</button>
```

Demo 8

The Vue Instance

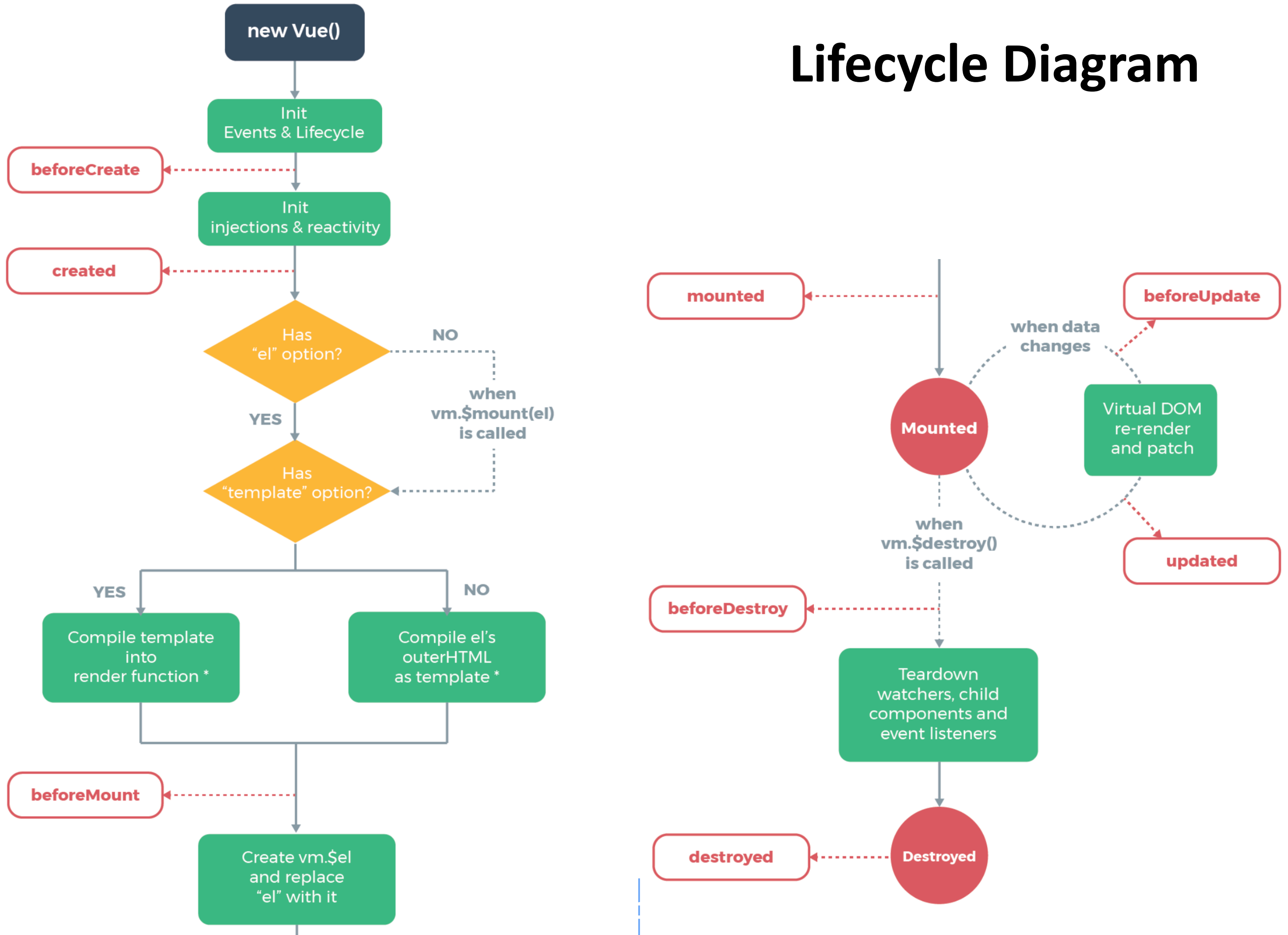
REliable, INtelligent & Scalable Systems

- Instance Lifecycle Hooks

```
Vue.createApp({
    data() {
        return{
            a: 1
        }
    },
    created() {
        // `this` points to the vm instance
        console.log('a is: ' + this.a)
    }
}).mount('#app') // => "a is: 1"
```

Demo 9

Lifecycle Diagram



new Vue()
Init Events & Lifecycle
beforeCreate
Init injections & reactivity
created
Has "el" option?
NO
YES
when vm.$mount(el) is called
Has "template" option?
YES
NO
Compile template into render function *
Compile el's outerHTML as template *
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
updated
when vm.$destroy() is called
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

Template Syntax

- Vue.js uses an HTML-based template syntax
  - that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data.
- All Vue.js templates are valid HTML
  - that can be parsed by spec-compliant browsers and HTML parsers.
- Under the hood, Vue compiles the templates into Virtual DOM render functions.
  - Combined with the reactivity system, Vue is able to intelligently figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

Template Syntax - Interpolations

- Text

```
<span>Message: {{ msg }}</span>
<span v-once>This will never change: {{ msg }}</span>
```

- RawHTML

```
<p>Using mustaches: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

- Attribute

```
<div v-bind:id="dynamicId"></div>
<button v-bind:disabled="isButtonDisabled">Button</button>
```

- Using JavaScript Expression

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO’ }}
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id"></div>
```

Demo 10

Template Syntax - Directives

- Arguments

```
<a v-bind:href="url"> ... </a>
<a v-on:click="doSomething"> ... </a>
```

- Dynamic Arguments

```
<a v-bind:[attributeName]="url"> ... </a>
<a v-on:[eventName]="doSomething"> ... </a>
```

- Modifier

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

Template Syntax - Shorthands

- v-bind Shorthand

```
<!-- full syntax -->
<a v-bind:href="url"> ... </a>
<!-- shorthand -->
<a :href="url"> ... </a>
<!-- shorthand with dynamic argument (2.6.0+) -->
<a :[key]="url"> ... </a>
```

- v-on Shorthand

```
<!-- full syntax -->
<a v-on:click="doSomething"> ... </a>
<!-- shorthand -->
<a @click="doSomething"> ... </a>
<!-- shorthand with dynamic argument (2.6.0+) -->
<a @[event]="doSomething"> ... </a>
```

Computed Properties

REliable, INtelligent & Scalable Systems

- Basic Example

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>

Vue.createApp({
    data(){
        return{
            message: 'Hello'
        }
    },
    computed: {
        // a computed getter
        reversedMessage () {
            // `this` points to the vm instance
            return this.message.split('').reverse().join('')
        }
    }
}).mount('#example')
```

Demo 11

Computed Properties

REliable, INtelligent & Scalable Systems

- Computed Caching vs Methods
  - You may have noticed we can achieve the same result by invoking a method in the expression:

```
<p>Reversed message: "{{ reverseMessage() }}"</p>
// in component
methods: {
  reverseMessage() {
    return this.message.split('').reverse().join(’’)
  }
}
```

  - This also means the following computed property will never update, because Date.now() is not a reactive dependency:

```
computed: {
  now() { return Date.now() }
}
```

Demo 12

Computed Properties

- Computed Setter

```
// ...
computed: {
Vue.createApp({
    data() {
        return {
            firstName: 'Foo',
            lastName: 'Bar',
        }
    },
    computed: {
        fullName: {
            // getter
            get() {
                return this.firstName + ' ' + this.lastName
            },
            // setter
            set(newValue) {
                // Note: we are using destructuring assignment syntax here.
                [this.firstName, this.lastName] = newValue.split(' ')
            }
        }
    }
}).mount('#demo')
```

Demo 13

Computed Properties

- Watcher

```
Vue.createApp({
  data() {
    return{
        question: ‘’,
        answer: ‘I cannot give you an answer until you ask a question!’
    }
  },
  watch: {
    question(newQuestion, oldQuestion) {
      this.answer = 'Waiting for you to stop typing...'
      this.debouncedGetAnswer()
    }
  },
  created() {
    this.debouncedGetAnswer = _.debounce(this.getAnswer, 500)
  },
```

Demo 14

Computed Properties

- Watcher

```
methods: {
  getAnswer() {
    if (this.question.indexOf(‘?’) === -1) {
      this.answer = ‘Questions usually contain a question mark. ;-)’
     return
    }
    this.answer = ‘Thinking...’
    var vm = this
    axios.get(‘https://yesno.wtf/api’)
       .then(function (response) {
          vm.answer = _.capitalize(response.data.answer)
        })
        .catch(function (error) {
          vm.answer = ‘Error! Could not reach the API. ’ + error
        })
   }
  }
})
```

Demo 14

Class and Style Bindings

- A common need for data binding is manipulating an element’s class list and its inline styles.

- Since they are both attributes, we can use v-bind to handle them:
  - we only need to calculate a final string with our expressions.
  - However, meddling with string concatenation is annoying and error-prone.
  - For this reason, Vue provides special enhancements when v-bind is used with class and style.
  - In addition to strings, the expressions can also evaluate to objects or arrays.

Binding HTML Classes

- Object Syntax

```
<div v-bind:class="{ active: isActive }"></div>

<div
  class="static"
  v-bind:class="{ active: isActive, 'text-danger': hasError }" >
</div>
data() {
  return { isActive: true, hasError: false }
}
```

=>

```
<div class="static active"></div>
```

Binding HTML Classes

REliable, INtelligent & Scalable Systems

- Object Syntax

```
<div v-bind:class="classObject"></div>
data() {
 return {
  classObject: {
   active: true,
   'text-danger': false
  }
 }
}
```

=>

```
<div class="static active"></div>
```

Binding HTML Classes

REliable, INtelligent & Scalable Systems

- Object Syntax

```
<div v-bind:class="classObject"></div>

data() {
  return {
    isActive: true,
    error: null
  }
},
computed: {
  classObject() {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal'
    }
  }
}
```

Binding HTML Classes

- Array Syntax

```
<div :class="[activeClass, errorClass]"></div>
data() {
  return{
    activeClass: 'active',
    errorClass: 'text-danger'
  }
}
```

=>

```
<div class="active text-danger"></div>
```

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>
<div v-bind:class="[{ active: isActive }, errorClass]"></div>
```

Binding HTML Classes

- With Components

```
Vue.component('my-component', {
  template: '<p class="foo bar">Hi</p>'
})
<my-component class="baz boo"></my-component>
```

=>

```
<p class="foo bar baz boo">Hi</p>
```

```
<my-component v-bind:class="{ active: isActive }"></my-component>
```

=>

```
<p class="foo bar active" >Hi</p>
```

Binding Inline Styles

- Object Syntax

```
<div
   v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">
</div>
data() {
   return { activeColor: 'red', fontSize: 30 }
}
```

```
<div v-bind:style="styleObject"></div>
data () {
  return {
    styleObject: {color: 'red', fontSize: '13px' }
  }
}
```

Binding Inline Styles

REliable, INtelligent & Scalable Systems

- Array Syntax

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

- Auto-prefixing
  - Vue will automatically detect and add appropriate prefixes to the applied styles

- Multiple Values

```
<div
v-bind:style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }">
</div>
```

Conditional Rendering

- v-if

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

- Conditional Groups with v-if on <template>

```
<template v-if="ok">
 <h1>Title</h1>
 <p>Paragraph 1</p>
 <p>Paragraph 2</p>
</template>
```

Conditional Rendering

- v-else

```
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don’t
</div>
```

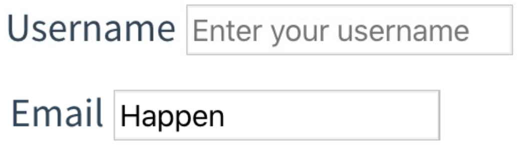
- v-else-if

```
<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> Not A/B/C </div>
```

Conditional Rendering

- Controlling Resusable Elements with key

```
<template v-if="loginType === 'username'">
  <label>Username</label>
  <input placeholder="Enter your username">
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Enter your email address">
</template>
```

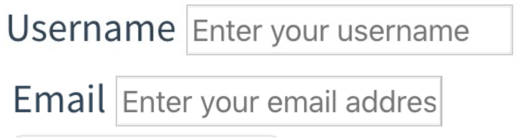

Username Enter your username
Email Happen

Demo 15

Conditional Rendering

- Controlling Resusable Elements with key

```
<template v-if="loginType === 'username'">
  <label>Username</label>
  <input placeholder="Enter your username" key= "username-input" >
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Enter your email address" key= "email-input" >
</template>
```



Conditional Rendering

- v-show

```
<h1 v-show="ok">Hello!</h1>
```

- v-if vs. v-show
  - v-if is “real” conditional rendering because it ensures that event listeners and child components inside the conditional block are properly destroyed and re-created during toggles.
  - v-if is also **lazy**: if the condition is false on initial render, it will not do anything - the conditional block won’t be rendered until the condition becomes true for the first time.
- v-if with v-for
  - Using v-if and v-for together is **not recommended**.

List Rendering

REliable, INtelligent & Scalable Systems

- Mapping an Array to Elements with v-for

```
<ul id="example-1">
 <li v-for="item in items">
  {{ item.message }}
 </li>
</ul>

Vue.createApp({
  data() {
    return{
      items: [
        { message: 'Foo' },
        { message: 'Bar' }
      ]
    }
  }
}).mount('#example-1')
```

List Rendering

- Mapping an Array to Elements with v-for

```
<ul id="example-1">
 <li v-for="item in items">
  {{ parentMessage }} - {{ index }} - {{ item.message }}
 </li>
</ul>

Vue.createApp({
    data() {
        return{
            parentMessage: 'Parent',
            items: [
                { message: 'Foo' },
                { message: 'Bar' }
            ]
        }
    }
}).mount('#example-1')
```

Demo 16

List Rendering

- v-for with an Object

```
<ul id="example">
 <li v-for= "(value, name, index) in object">
   {{ index }} . {{ name }} : {{ value }}
 </li>
</ul>


Vue.createApp({
    data() {
        return{
            object: {
                title: 'How to do lists in Vue',
                author: 'Jane Doe',
                publishedAt: '2016-04-10'
            }
        }
    }
}).mount('#v-for-object')
```

Demo 17

Array Change Detection

- Mutation Methods
  - Vue wraps an observed array's mutation methods so they will also trigger view updates. The wrapped methods are:
  - `push()`
  - `pop()`
  - `shift()`
  - `unshift()`
  - `splice()`
  - `sort()`
  - `reverse()`

  - Demo 4: `app4.todos.push({text: 'Learn React'})`

Array Change Detection

REliable, INtelligent & Scalable Systems

- Replacing an Array

```
example1.items = example1.items.filter(function (item) {
 return item.message.match(/Foo/)
})
```

  - Demo 4:

```
app4.todos = app4.todos.filter(function (item) {
  return item.text.match('Learn Vue')
})
```

Array Change Detection

- Caveats
  - Due to limitations in JavaScript, Vue **cannot** detect the following changes to an array:
  - When you directly set an item with the index, e.g.
    `vm.items[indexOfItem] = newValue`
  - When you modify the length of the array, e.g.
    `vm.items.length = newLength`

```
var vm = new Vue({
 data: {
   items: ['a', 'b', 'c'] }
})
vm.items[1] = 'x' // is NOT reactive
vm.items.length = 2 // is NOT reactive
```

Array Change Detection

- Caveats

```
var vm = new Vue({
 data: {
   items: ['a', 'b', 'c'] }
})
vm.items[1] = 'x' // is NOT reactive
```

=>

```
// Vue.set
Vue.set(vm.items, indexOfItem, newValue)
// Array.prototype.splice
vm.items.splice(indexOfItem, 1, newValue)
// instance.set
vm.$set(vm.items, indexOfItem, newValue)
```

Array Change Detection

- Caveats

```
var vm = new Vue({
 data: {
   items: ['a', 'b', 'c'] }
})
vm.items.length = 2 // is NOT reactive
```

=>

```
vm.items.splice(newLength)
```

Array Change Detection

- Object Change Detection Caveats
  - Vue cannot detect property addition or deletion.

```
var vm = new Vue({
  data: {
   a: 1
  }
}) // `vm.a` is now reactive

vm.b = 2 // `vm.b` is NOT reactive
```

  - Vue does not allow dynamically adding new root-level reactive properties to an already created instance.

Array Change Detection

REliable, INtelligent & Scalable Systems

- Object Change Detection Caveats
  - However, it’s possible to add reactive properties to a nested object.

```
var vm = new Vue({
  data: {
   userProfile: {
    name: 'Anika'
  }
})
```

```
– Vue.set(vm.userProfile, 'age', 27)
– vm.$set(vm.userProfile, 'age', 27)
– Object.assign(vm.userProfile, {
     age: 27,
     favoriteColor: 'Vue Green’
  })
– vm.userProfile = Object.assign({}, vm.userProfile, {
     age: 27,
     favoriteColor: 'Vue Green’
  })
```

Array Change Detection

- Displaying Filtered/Sorted Results

```
<li v-for="n in evenNumbers">{{ n }}</li>

data() {
    return{
        numbers: [ 1, 2, 3, 4, 5 ]
    }
},
computed: {
    evenNumbers: function () {
        return this.numbers.filter(function (number) {
            return number % 2 === 0
        })
    }
}
```

Demo 18

Array Change Detection

- Displaying Filtered/Sorted Results

```
<ul v-for="set in sets">
 <li v-for="n in even(set)">{{ n }}</li>
</ul>

Vue.createApp({
    data() {
        return{
            sets: [[ 1, 2, 3, 4, 5 ], [6, 7, 8, 9, 10]]
        }
    },
    methods: {
        even: function (numbers) {
            return numbers.filter(function (number) {
                return number % 2 === 0
            })
        }
    }
}).mount('#object')
```

Demo 19

Array Change Detection

- v-for with a Range

```
<div> <span v-for="n in 10">{{ n }} </span> </div>
```

Array Change Detection

REliable, INtelligent & Scalable Systems

- v-for with a Component

```
<div id="todo-list-example">
 <form v-on:submit.prevent="addNewTodo">
  <label for="new-todo">Add a todo</label>
  <input
   v-model="newTodoText"
   id="new-todo"
   placeholder="E.g. Feed the cat"
  >
  <button>Add</button>
 </form>
 <ul>
  <todo-item
   is="todo-item"
   v-for="(todo, index) in todos"
   v-bind:key="todo.id"
   v-bind:title="todo.title"
   v-on:remove="todos.splice(index, 1)" >
  </todo-item>
 </ul>
</div>
```

Array Change Detection

REliable, INtelligent & Scalable Systems

- v-for with a Component

  todoitemx.js

```
export default {
    props: ['title'],
    template: '\
    <li>\
      {{ title }}\
      <button v-on:click="$emit(\'remove\')">Remove</button>\
    </li>\
    '
}
```

Array Change Detection

- v-for with a Component

```
import TodoItem from “./todoitemx.js”;

Vue.createApp({
    components: {
        TodoItem
    },
    data(){
        return{
          newTodoText: '',
          todos: [
             { id: 1, title: 'Do the dishes', },
             { id: 2, title: 'Take out the trash', },
             { id: 3, title: 'Mow the lawn' } ],
          nextTodoId: 4
        }
    },
    methods: {
        addNewTodo() {
            this.todos.push({
                id: this.nextTodoId++,
                title: this.newTodoText
            })
            this.newTodoText = ''
        }
    }
}).mount('#todo-list-example')
```

Demo 20

Event Handling

- Listening to Events

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>

Vue.createApp({
  data() {
    return{
      counter: 0
    }
  }
}).mount('#example-1')
```

Event Handling

- Method Event Handlers

```
<div id="example-3">
  <button v-on:click="say('hi')">Say hi</button>
  <button v-on:click="say('what')">Say what</button>
</div>

Vue.createApp({
  methods: {
    say(message) {
      alert(message)
    }
  }
}).mount('#example-3')
```

Event Handling

REliable, INtelligent & Scalable Systems

- Method Event Handlers

```
<button v-on:click=
       "warn('Form cannot be submitted yet.', $event)">
  Submit
</button>

methods: {
  warn (message, event) {
    // now we have access to the native event
    if (event) {
      event.preventDefault()
    }
    alert(message)
  }
}
```

Event Handling

- Event Modifiers

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- modifiers can be chained -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- just the modifier -->
<form v-on:submit.prevent></form>

<!-- use capture mode when adding the event listener -->
<!-- i.e. an event targeting an inner element is handled here -->
<!-- before being handled by that element -->
<div v-on:click.capture="doThis">...</div>
```

Event Handling

- Event Modifiers

```
<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>

<!-- the click event will be triggered at most once -->
<a v-on:click.once="doThis"></a>

<!-- the scroll event's default behavior (scrolling) will happen -->
<!-- immediately, instead of waiting for `onScroll` to complete -->
<!-- in case it contains `event.preventDefault()` -->
<div v-on:scroll.passive="onScroll">...</div>
```

Event Handling

- Key Modifiers

```
<!-- only call `vm.submit()` when the `key` is `Enter` -->
<input v-on:keyup.enter="submit">

<input v-on:keyup.page-down="onPageDown">
```

Key Code
- .enter, .tab, .delete (captures both “Delete” and “Backspace” keys)
- .esc, .space, .up, .down, .left, .right

```
<input v-on:keyup.13="submit">
```

Event Handling

- System Modifier Keys
  - .ctrl, .alt, .shift, .meta

```
<!-- Alt + C -->
<input v-on:keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div v-on:click.ctrl="doSomething">Do something</div>
```

.exact Modifer

```
<!-- this will fire even if Alt or Shift is also pressed -->
<button v-on:click.ctrl="onClick">A</button>

<!-- this will only fire when Ctrl and no other keys are pressed -->
<button v-on:click.ctrl.exact="onCtrlClick">A</button>

<!-- this will only fire when no system modifiers are pressed -->
<button v-on:click.exact="onClick">A</button>
```

Form Input Bindings – Basic Usage

REliable, INtelligent & Scalable Systems

- Text

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

- Multiline text

```
<span>Multiline message is:</span>
<p style="white-space: pre-line;">{{ message }}</p>
<br>
<textarea v-model="message" placeholder="add multiple lines">
</textarea>
```

- Checkbox

```
<input type="checkbox" id ="checkbox" v-model ="checked" >
<label for ="checkbox">{{ checked }}</label>
```

Demo 21

Form Input Bindings – Basic Usage

- Checkbox

```
<div id='example-3'>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames">
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
  <label for="mike">Mike</label>
  <br>
  <span>Checked names: {{ checkedNames }}</span>
</div>

Vue.createApp({
  data() {
    return{
      checkedNames: []
    }
  }
}).mount('#example-3')
```

Demo 21

Form Input Bindings – Basic Usage

- Radio

```
<input type="radio" id="one" value="One" v-model="picked">
<label for="one">One</label>
<br>
<input type="radio" id="two" value="Two" v-model="picked">
<label for="two">Two</label>
<br>
<span>Picked: {{ picked }}</span>
```

Demo 21

Form Input Bindings – Basic Usage

- Select – Single Select

```
<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<span>Selected: {{ selected }}</span>

Vue.createApp({
  data: {
    return{
      selected: ''
    }
  }
}).mount('#example-3')
```

Demo 21

Form Input Bindings – Basic Usage

- Select – Multiple Select

```
<select v-model="selected" multiple>
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<span>Selected: {{ selected }}</span>

Vue.createApp({
  data: {
    return{
      selected: ''
    }
  }
}).mount('#example-3')
```

Demo 21

Form Input Bindings – Basic Usage

REliable, INtelligent & Scalable Systems

- Select – Dynamic options rendered with v-for

```
<select v-model="selected">
   <option v-for="option in options" v-bind:value="option.value">
     {{ option.text }}
   </option>
</select>
<span>Selected: {{ selected }}</span>

Vue.createApp({
  data() {
    return{
       selected: 'A',
       options: [
         { text: 'One', value: 'A' },
         { text: 'Two', value: 'B' },
         { text: 'Three', value: 'C' } ]
    }
  }
}).mount('#example-3')
```

Demo 21

Form Input Bindings – Value Binding

- Value Binding
  - For radio, checkbox and select options, the v-model binding values are usually static strings (or booleans for checkbox):

```
<!-- `picked` is a string "a" when checked -->
<input type="radio" v-model="picked" value="a">

<!-- `toggle` is either true or false -->
<input type="checkbox" v-model="toggle">

<!-- `selected` is a string "abc" when the first option is selected -->
<select v-model="selected">
  <option value="abc">ABC</option>
</select>
```

  - But sometimes we may want to bind the value to a dynamic property on the Vue instance.
  - We can use v-bind to achieve that.
  - In addition, using v-bind allows us to bind the input value to non-string values.

Demo 22

Form Input Bindings – Value Binding

REliable, INtelligent & Scalable Systems

- Checkbox

```
<input
  type="checkbox"
  v-model="toggle"
  true-value="yes"
  false-value="no"
>

// when checked:
vm.toggle === 'yes'
// when unchecked:
vm.toggle === 'no'
```

Demo 22

Form Input Bindings – Value Binding

- Radio

```
<input
  type="radio"
  v-model="pick"
  v-bind:value="a"
>

// when checked:
vm.pick === a
```

Demo 22

Form Input Bindings – Value Binding

- Select options

```
<select v-model="selected">
  <!-- inline object literal -->
  <option v-bind:value="{ number: 123 }">123</option>
</select>

// when selected:
typeof vm.selected // => 'object’
vm.selected.number // => 123
```

Demo 22

Form Input Bindings – Modifier

- .lazy

```
<!-- synced after "change" instead of "input" -->
<input v-model.lazy="msg">
```

- .number

```
<input v-model.number="age" type="number">
```

- .trim

```
<input v-model.trim="msg">
```

Demo 22

Component Basic

- Base Example

```
<div id="components-demo">
  <button-counter></button-counter>
</div>

<script>
  // Define a new component called button-counter
  Vue.component('button-counter', {
     data: function () {
         return {
             count: 0
         }
     },
     template: '<button v-on:click="count++">
        You clicked me {{ count }} times.</button>'
  })
  new Vue({ el: '#components-demo' })
</script>
```

Demo 23

Component Basic

- Reusing Components

```
<div id="components-demo">
  <button-counter></button-counter>
  <button-counter></button-counter>
  <button-counter></button-counter>
</div>
```

- data Must Be a Function

```
data: function () {
    return {
        count: 0
    }
},
```

Demo 23

Component Basic

- Passing Data to Child Components with Props

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>‘
})

<div id="blog-demo">
  <blog-post title="My journey with Vue"></blog-post>
  <blog-post title="Blogging with Vue"></blog-post>
  <blog-post title="Why Vue is so fun"></blog-post>
</div>

new Vue({el: '#blogs-demo'})
```

Demo 23

Component Basic

- Passing Data to Child Components with Props

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>‘
})

 <div id="blog-post-demo">
     <blog-post
             v-for="post in posts"
             v-bind:key="post.id"
             v-bind:title="post.title"
     ></blog-post>
 </div>

 new Vue({
     el: '#blog-post-demo',
     data: {
         posts: [
             {id: 1, title: 'My journey with Vue'},
             {id: 2, title: 'Blogging with Vue'},
             {id: 3, title: 'Why Vue is so fun'}
         ]
     }
 })
```

Demo 23

Component Basic

- A Single Root Element
  - Every component must have a single root element.

```
<div class="blog-post">
  <h3>{{ title }}</h3>
  <div v-html="content"></div>
</div>

<blog-post  // error-prone
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:title="post.title"
  v-bind:content="post.content"
  v-bind:publishedAt="post.publishedAt"
  v-bind:comments="post.comments"
></blog-post>
```

Component Basic

- A Single Root Element
  - Every component must have a single root element.

```
<blog-post
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:post="post"
></blog-post>

Vue.component('blog-post', {
  props: ['post'],
  template: `
    <div class="blog-post">
      <h3>{{ post.title }}</h3>
      <div v-html="post.content"></div>
    </div> `
})
```

Component Basic

REliable, INtelligent & Scalable Systems

- Listening to Child Component Events

```
Vue.component('blog-post-event', {
    props: ['post'],
    template: '
      <div class="blog-post">
        <h3>{{ post.title }}</h3>
        <button v-on:click="$emit(\'enlarge-text\')">
           Enlarge text
        </button>
        <div v-html="post.content"></div>
     </div>’
})
```

Demo 23

Component Basic

- Listening to Child Component Events

```
<div id="blog-posts-events-demo">
    <div :style="{ fontSize: postFontSize + 'em' }">
        <blog-post-event
                v-for="post in posts"
                v-bind:key="post.id"
                v-bind:post="post"
                v-on:enlarge-text="postFontSize += 0.1"
        ></blog-post-event>
    </div>
</div>

new Vue({
    el: '#blog-posts-events-demo',
    data: {
        posts: [
            {id: 1, title: 'My journey with Vue'},
            {id: 2, title: 'Blogging with Vue'},
            {id: 3, title: 'Why Vue is so fun'}
        ],
        postFontSize: 1
    }
})
```

Demo 23

Component Basic

- Emitting a Value With an Event

```
Vue.component('blog-post-event', {
    props: ['post'],
    template: '
      <div class="blog-post">
        <h3>{{ post.title }}</h3>
        <button v-on:click="$emit(\'enlarge-text\', 0.1)">
           Enlarge text
        </button>
        <div v-html="post.content"></div>
     </div>’
})
```

Component Basic

- Then when we listen to the event in the parent, we can access the emitted event's value with $event:

```
<blog-post ... v-on:enlarge-text="postFontSize += $event" >
</blog-post>
```

- Or, if the event handler is a method:

```
<blog-post ... v-on:enlarge-text="onEnlargeText" ></blog-post>
```

- Then the value will be passed as the first parameter of that method:

```
methods: {
 onEnlargeText: function (enlargeAmount) {
     this.postFontSize += enlargeAmount
 }
}
```

Component Basic

REliable, INtelligent & Scalable Systems

- Using v-model on Components

```
<input v-model="searchText">
=>
<input
  v-bind:value="searchText"
  v-on:input="searchText = $event.target.value"
>
```

```
Vue.component('custom-input’, {
  props: ['value’],
  template: `
    <input
      v-bind:value="value"
      v-on:input="$emit('input', $event.target.value)" > `
})
<custom-input v-model="searchText"></custom-input>
```

Demo 23

Component Basic

REliable, INtelligent & Scalable Systems

- Content Distribution with Slots

```
<alert-box> Something bad happened. </alert-box>

Vue.component('alert-box', {
  template: `
    <div class="demo-alert-box">
      <strong>Error!</strong>
      <slot></slot>
    </div> `
})
```

Demo 23

- *Web开发技术*
- *Web Application Development*

Thank You!