Vue In 30 Minutes

Binghua Wu, 8 Feb 2017

What is Vue?

- **Vue** is a JavaScript framework.
- Pronounced `/vjuː/`, like ***view***
- Created by *Yuxi You*
- MIT License

Rendering Performance Compare to Reactive

	Vue	**React**
Fastest	23ms	63ms
Median	42ms	81ms
Average	51ms	94ms
95th Perc.	73ms	164ms
Slowest	343ms	453ms

# See Benchmark

Rendering Performance Compare with Other Frameworks

JS Framework Benchmark

Create a View Model

```
var vm = new Vue({
  // options
})
```

or

```
var MyComponent = Vue.extend({
  // extension options
})
// all instances of `MyComponent` are created with
// the pre-defined extension options
var myComponentInstance = new MyComponent()
```

Instance Lifecycle

https://vuejs.org/images/lifecycle.png

Instance Lifecycle Hooks

```
var vm = new Vue({
  data: {
    a: 1
  },
  // called after the component is created.
  created: function () {
    // `this` points to the vm instance
    console.log('a is: ' + this.a)
  }
})
// -> "a is: 1"
```

Template Syntax -- Directives

Directives are special attributes with the `v-` prefix.

```
<p v-if="seen">Now you see me</p>
<!-- bind the url to href attribute of tag 'a' -->
<a v-bind:href="url"></a>
<!-- apply filters to format text value, filter functions are denoted by 'pipe' s
ymbol -->
<div v-bind:id="rawId | formatId"></div>
<!-- filters can be chained -->
<a v-bind:url>{{ url | replacePlaceholder | encode }}</p>
<!-- shorthand for v-bind -->
<a :href="url"></a>
<!-- shorthand for v-on -->
<a @click="doSomething"></a>
```

Template Syntax -- Computed Properties

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

```js
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      return this.message.split('').reverse().join('')
    }
  }
})
```

Template Syntax -- Watched Properties

Vue provides a more generic way to react to data changes through the `watch` option.

```
var watchExampleVM = new Vue({
  ...
  data: {
    question: '',
    answer: 'I cannot give you an answer until you ask a question!'
  },
  watch: {
    // watching "question" property, whenever question changes, this function wil
l run
    question: function (newQuestion) {
      this.answer = 'Waiting for you to stop typing...'
      this.getAnswer()
    }
  }
...
})
```

Template Syntax -- Computed vs. Watched Properties

- Computed property value will be cached to improve rendering performance.
- Watch function is computing heavy
- Watch function is suitable for asynchronous call

Class and Style Binding

```
<div class="static"
     v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
```

```
data: {
  isActive: true,
  hasError: false
}
```

Will render:

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

Class and Style Binding

Inline style binding

```
<div v-bind:style="styleObject"></div>
```

```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

Output:

```
<div style="color:red, fontSize:13px"></div>
```

Conditional Rendering

```
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
<h1 v-show="ok">Hello!</h1>
```

- `v-if` is lazy rendering, won't render until condition meets
- `v-show` elements always get rendered, toggle show or hide in the CSS display way
- `v-show` suitable for scenario that toggle something very often

List Rendering

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

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

Two-way Binding

Use the `v-model` directive to create two-way data bindings

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

Components

Coponents are basic reusable HTML code wrapped by Vue

Create a Component:

```
Vue.component('my-component', {
  // options
})
```

Use the component:

```
<div id="example">
  <my-component></my-component>
</div>
```

Components

In case custome tag name is forbidden, use `is` attribute.

```html
<table>
  <tr is="my-row"></tr>
</table>
```

Components

`data` must be function

```
Vue.component('simple-counter', {
  template: '<button v-on:click="counter += 1">{{ counter }}</button>',
  // data: {counter: 0} // this is wrong
  data: function () {
    return {counter: 0};
  }
})
```

Components -- Props

- An option in component constructor
- Data can be passed down to child components using `props`
- Use camelCased prop name in JavaScript, use hyphen-delimited name in HTML
- Prop validation is supported

Components -- Props

```
Vue.component('example', {
  props: {
    // basic type check (`null` means accept any type)
    propA: Number,
    // multiple possible types
    propB: [String, Number],
    // a number with default value
    propD: {
      type: Number,
      required: true,
      default: 100
    },
    // object/array defaults should be returned from a factory function
    propE: {
      type: Object,
      default: function () {
        return { message: 'hello' }
      }
    },
    // custom validator function
    propF: {
      validator: function (value) {
        return value > 10
      }
    }
  }
})
```

Components -- Custom Events

- Listen to an event using `$on(eventName)`
- Trigger an event using `$emit(eventName)`

```html
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter v-on:increment="incrementTotal"></button-counter>
</div>
```

```javascript
Vue.component('button-counter', {
  template: '<button v-on:click="increment">{{ counter }}</button>',
  data: function () {
    return {
      counter: 0
    }
  },
  methods: {
    increment: function () {
      this.counter += 1
      this.$emit('increment')
    }
  },
})
```

Reference

- User Guide
- API

bhpbilliton