Vue.js - all you can do

Intro and Case Studies

Olga Khorkova

Developer Open Space

13. November 2020

Inhalte

1. General
   - About Vue
   - Characteristics
2. Basics: DOM Interactions
3. Advanced Vue
   - Components
   - Communication between Components
   - Lifecycle Hooks
   - Routing
   - Vuex
   - Data Handling
4. Vue3/Vue2
   - Changed
   - New
5. Case Studies

Agenda

- Start 9:15 a.m.
- Lunch 12 - 13 p.m.
- End 16 p.m.
- Short breaks (10-15 min) after each block

Purpose of the day

By the end of the day, each participant should be able to re-think (and re-do) UI-Elements and large projects in terms of Vue.

About me

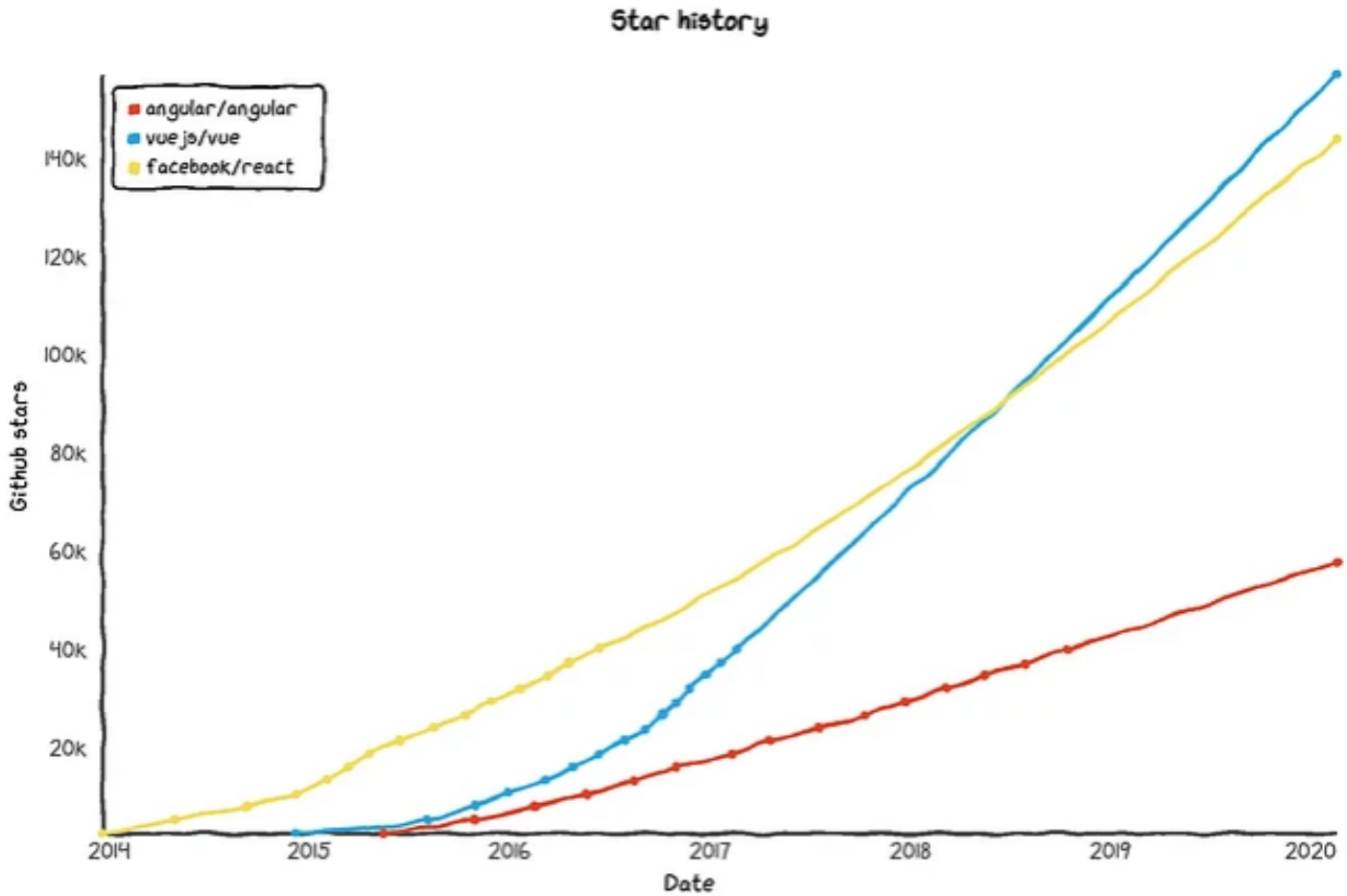
- Olga Khorkova
- Fullstack dev (JS/PHP)
- Twitter @prolga_

About Vue

Vue

1. open-source JS framework/library
2. 20KB min+gzip
3. could be applied for both UI and SPA
4. written by Evan You
   `https://www.youtube.com/watch?v=OrxmtDw4pVI`
5. huge Chinese community

Vue Popularity



source: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Vue3/Vue2

Last stable release 3.0.2 / October 20, 2020. However, not all ecosystem adopted to new syntax/new features.

> All of our official libraries and tools now support Vue 3, but most of them are still in beta status and distributed under the `next` dist tag on NPM. **We are planning to stabilize and switch all projects to use the `latest` dist tag by end of 2020.**

https://github.com/vuejs/vue-next

Vue3/Vue2

What should I learn?

Vue3/Vue2

→ **Vue3**

Vue is declarativ

1. enables us to declaratively render data to the DOM using simple, straightforward template syntax {{*var*}}
2. compare to vanilla JS `document.createElement(''LI'');`

Vue is reactiv

1. enables immediately react to changes (one or two-ways-data-binding)
2. behind the scenes - Proxy()
3. it is impossible in vanilla JS

Vue is reactiv: Question

What are the outputs?

```
1  var a = 1;
2  var b = 2;
3
4  var sum = a + b;
5  console.log(sum);
6
7  var a = 6;
8  console.log(sum);
9
```

Vue: Virtual DOM

Virtual DOM - representation of DOM nodes as JS data structure `https://codepen.io/sdras/full/RwwQapa`

image source: https://vuejsdevelopers.com/2017/02/21/vue-js-virtual-dom/

Vue is component-based

1. in general, *each structural part of an app is a component*
2. i.e layouts, pages, block etc - all of these are components
3. similar logic to HTML/Web Components `https://developer.mozilla.org/en-US/docs/Web/Web_Components`
4. the best way to ensure modularity and encapsulation

Basics

Basic Vue Instance

Library is imported as CDN package

JS

```
// Create a Vue application
const app = Vue.createApp({
  data() {
    return {
      counter
    }
  },
  methods: {
    increaseCounter () {
     this.counter++
    }
  }
})

app.mount('#app')
```

HTML

```
<main id="app">
  {{counter}}
  <button v-on:click="increaseCounter">more</button>
</main>
```

Basics

1. Vue Syntax starts with `v-`
2. There is a data method that stores data used inside current component
3. It can contain methods
4. The HTML-template selected by id is mounted on the app

Interpolation

1. HTML-Syntax $\{\{var\}\}$
2. Vue renders code between brackets to DOM
3. Acessible data: properties and methods

Binding Attributes

1. HTML-Atirbutes could be bind dynamically
2. Syntax `v-bind:class=''{active: true}''`
3. Shortcut `:class=''{active: true}''`
4. More attributes?

Special Case: in order to get html tags rendered use `v-html` directive

Binding Events

1. We can listen to (native or custom) events happenning on certain elements
2. Syntax `v-on:click=''counter++''`
3. Shortcut `@click=''counter++''`
4. Events can have modifier`@:submit.prevent=''sendAnswer''`
5. More events?

Exercise 1

→ **Exercise 1**

Methods

1. Methods are accessible *data* in the template
2. Syntax `{{increaseCounter()}}`
3. or `@click=''increaseCounter''`

Forms

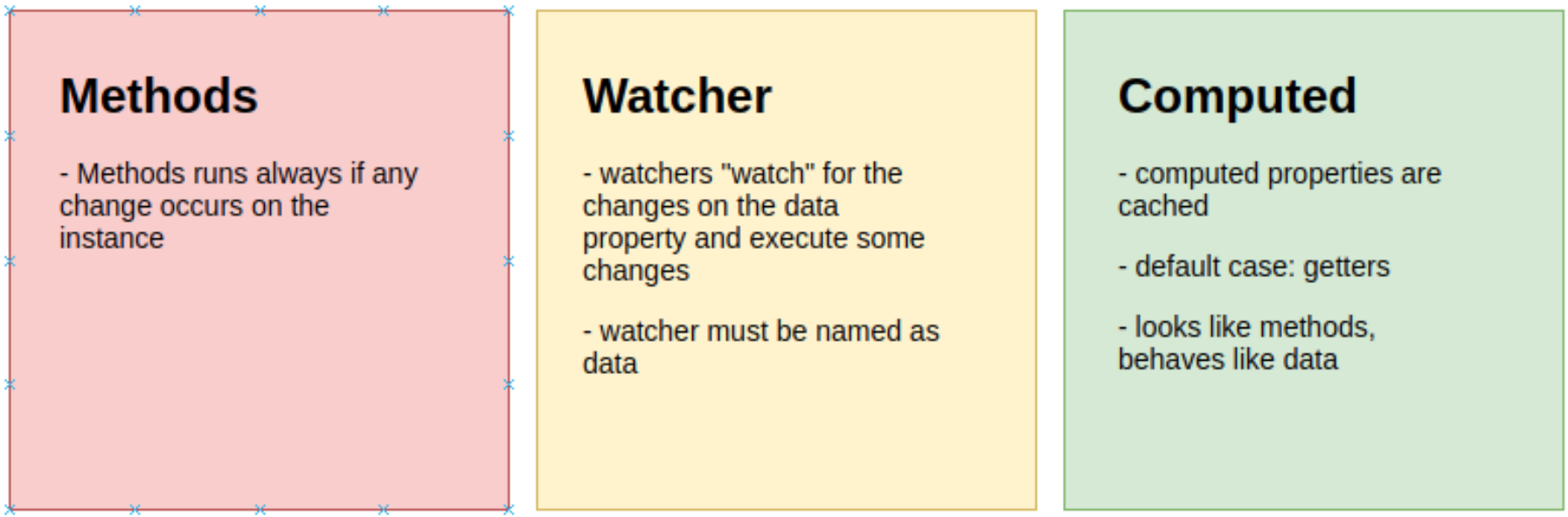
1. Forms submit data
2. Syntax `<form @submit.prevent=''sendSomeData''>`
3. In most cases, input elements have binding atrribute `<input v-model=''name''>` which (a) has two-way-binding and (b) reacts to the changes
4. However, one can listen to events on them `<input @input=''alert''>`

Exercise 2

→ **Exercise 2**

Alternative to Methods

Methods are not perfomant if they are used directly in template for data modification



Computed/Watch: Syntax

```
data() {
  return {
    year: 2020,
    name: 'Kamala',
    surName: 'Harris'
  }
},
method: {
  increaseYear() {
    this.year += 4
  }
},
watch: {
  year() {
    if (year === 2020) {
      console.log('Joe')
    } else {
      console.log('Donald')
    }
  }
},
computed: {
  fullName() {
    return this.name + ' ' + this.surName
  }
}
})
```

Conditional Rendering

1. Syntax
   1. `v-if=''active === 'green''`
   2. `v-else=''active === 'blau''`
   3. `v-else-if=''active === 'true''`
2. Often used inside `<template>`

List Rendering

1. Syntax `<li v-for="item in items" :key="item.id">{{item.title}}</li>`
2. Cannot be used inside `<template>`

Styling

1. Inline styling `:style=''borderColor: 'red'''`
2. Dynamic classes `:class=''active: true''`

Exercise 3

→ **Exercise 3**

Components

Components: Structure

Syntax

JS

```
// Create a Vue application
const app = Vue.createApp({})

// Define a new global component called button-counter
app.component('button-counter', {
  data() {
    return {
      count: 0
    }
  },
  template: `
    <button v-on:click="count++">
      You clicked me {{ count }} times.
    </button>`
})

app.mount('#components-demo')
```

HTML

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

Exercise 4

→ **Exercise 4**

Components: @vue/cli

For large projects and better organization

→ `vue create knowledge-exchange`

Exercise 5

→ **Exercise 5**

Reusable components: slots

Slot is mechanism that allows to make components more generic i.e. give you an outlet to place content in new places

Parent

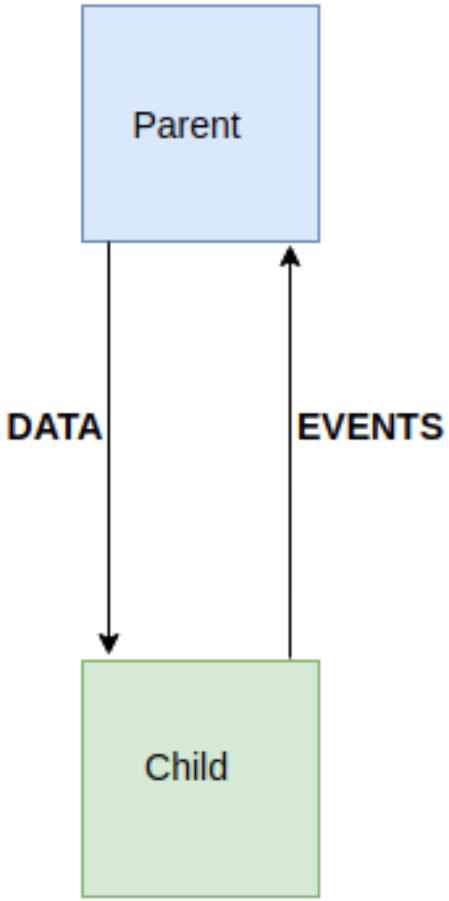
```
<alert>
    How many times do you want to try?
</alert>
```

Child

```
<template>
  <section class="block block-alert">
    <div>
      <slot />
    </div>
    <div>
      <button @click="close">close</button>
    </div>
  </section>
</template>
```

Data Flow

Data travels down, events – up!



props: syntax

Props Syntax:

Parent

```
<discussion-comment :comment="comment"/>
```

Child

```
<template>
  <div class="discussion-comment">
    <div class="discussion__comment">
      {{comment}}
    </div>
  </div>
</template>

<script>
export default {
  props: {
    comment: String
}}
</script>
```

props: semantics

1. Props are passed down the component tree to descendents (not up)
2. Props are read-only and cannot be modified

Custom events: syntax

Custom Events Syntax:

Parent

```
<alert v-if="alert!== ''" @close="alert = ''">
```

Child

```
<template>
  <section class="block block-alert">
      <div>
        Text
      </div>
      <div>
        <button @click="close">close</button>
      </div>
  </section>
</template>

<script>
export default {
  name: 'Alert',
  methods: {
    close() {
      this.$emit('close')
    }
  }
      You, a few seconds ago • Uncommitted
```

Exercise 6

→ **Exercise 6**

Lifecycle

1. Each component goes through the same lifecycle
2. They give ability to run code when a component reaches a particular state in execution

Lifecycle: Basic Hooks

The basic hooks are:

1. created - Called after the component has been created.
2. mounted - Called when the component has been mounted (browser updated).
3. updated - Called when reactive data has changed, and the DOM has been re-rendered.
4. destroyed - Called after the component has been destroyed.

Each of them has a "paring" hook prefixed by `before` → e.g. `beforeCreated()`

Lifecycle: Note Destroyed

→ **destroyed()/unmounted()**

Unmounting in Vue 3

In Vue 3 `beforeDestroy()` can also be written as `beforeUnmount()`, and `destroyed()` can be written as `unmounted()`. When I asked Evan You about these changes, he mentioned it's just a better naming conventions, because Vue *mounts* and *unmounts* components.

Routing

Vuex

Data Handling

→ axios

Changed Vue3 Features 1

1. The way how app is created: `createApp()`

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
}).$mount('#app')
```

Abbildung: Vue 2

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'

createApp(App).mount('#app')
```

Abbildung: Vue 3

Changed Vue3 Features 2

1. Components, directive, third-party are registered on app instead of Vue global object
2. data ist always a method
3. emittted events could be declared additionaly to props
4. Transitions: `v-enter` is now `v-enter-from`
5. The ways how to create Router and Store: `createRouter()` and `createStore()`

New Vue3 Features

1. Teleport: rendering in a different place (modals)
2. Multiple Root is possible
3. Composition API - optional
4. Better Typescript support

Case Studies

UI Components

1. forms (a bit outdated, just for comparison `https://www.smashingmagazine.com/2018/02/jquery-vue-javascript/`)
2. Mobile menu
3. Progressbar
4. Filters
5. Slides
6. etc.

Types of Vue Application

1. Client-Side
2. Server-Side (e.g. Nuxt.js)
3. Pre-rendered (Static) (e.g. Gridsome - JAM)

Danke!

Fragen? Vorschläge?