Vue.js reactivity fundamentals

WASA: Web and Software Architecture

Enrico Bassetti

Reactive programming

Reactive programming

Reactive programming is a **declarative** programming paradigm.

Two main ideas: **data streams**, and **propagation of change**.

Reactive vs imperative

In reactive languages, reactive variables are re-evaluated when one dependency changes.

```
reactive var b = a + 1
```

The reactive variable *b* changes when *a* changes.

Reactive vs imperative

Imperative languages:

```
var a = 2
var b = a + 1
fmt.Println(b) // Output: 3
a = 1
fmt.Println(b) // Output: 3
```

Reactive languages:

```
var a = 2
reactive var b = a + 1
fmt.Println(b) // Output: 3
a = 1
fmt.Println(b) // Output: 2
```

Links

That's all you need to know for reactive programming for this course.

If you want to know more:

- https://www-sop.inria.fr/mimosa/rp/generalPresentation/index.html
- https://xgrommx.github.io/rx-book/
- https://en.wikipedia.org/wiki/Reactive_programming

Vue.js reactivity

Reactive State

Each component in Vue.js has a *reactive state*.

```
<script>
export default {
  data() {
    // This is the reactive state
    return {
      count: 0,
    }
  }
}
</script>
<template><!-- ... --></template>
<style>/* ... */</style>
```

Reactive State

- Vue.js wraps the return of `data()` into its reactive system
- You must declare all reactive variables in `data()` return object
- Use `undefined` or `null` if there is no value
- Do not use reserved prefixes `$` or `_`
- Vue.js is deep-reactive: changes in nested objects are detected

Access the reactive state from JS

You can access the reactive variable from JS using *this*:

```
export default {
  data() {
    return { count: 1 }
  },
  // `mounted` is a lifecycle hook (we will see them later)
  mounted() {
    console.log(this.count) // => 1
    // you can change it like a normal variable
    // this will trigger the update in DOM and
    // all computed properties depending on it
    this.count = 2
  }
}
```

Access the reactive state from template

No need for *this* keyword in templates:

```
<span>Count: {{ count }}</span>
<div v-text="count"></div>
```

Variables are not pure

Note that variables from `data()` function are not "pure", they're wrapped in the reactive system:

```
export default {
  data() {
    return {
      someObject: {}
    }
  },
  mounted() {
    const newObject = {}
    this.someObject = newObject
    console.log(newObject === this.someObject) // false
  }
}
```

Computed properties

You can define *computed properties*. They are updated when their dependencies changes (reactivity).

```
<script>
export default {
  data() {
    return { count: 0 }
  },
  computed: {
    realCount() { return this.count + 1 }
  }
}
</script>
<template>
  <span>{{ realCount }}</span>
</template>
```

DOM reactivity

When you mutate reactive state, Vue.js updates the DOM *automatically*, but **not synchronously**.

Changes are **buffered** until the next “tick” (update cycle).

Access the DOM after “tick”

If you need access the DOM **after** the update cycle:

```
import { nextTick } from 'vue'
export default {
  // ...
  methods: {
    increment() {
      this.count++
      nextTick(() => {
        // access updated DOM
      })
    }
  }
}
```

Links

- https://vuejs.org/guide/essentials/reactivity-fundamentals.html
- https://vuejs.org/guide/extras/reactivity-in-depth.html

Component methods

Declaring methods

You can declare component methods, and call them from JS or the template itself.

```
export default {
  data() {
    return { count: 0 }
  },
  methods: {
    increment() { this.count++ }
  },
  // `mounted` is a lifecycle hook (we will see them later)
  mounted() {
    this.increment()
  }
}
```

Declaring methods: arrow notation (NO)

Do not define arrow functions:

```
export default {
  methods: {
    increment: () => {
      // BAD: no `this` access here!
    }
  }
}
```

Call methods in template

```
<script>
export default {
  data() {
    return { count: 0 }
  },
  methods: {
    increment() { this.count++ }
  }
}
</script>
<template>
  <button @click="increment">{{ count }}</button>
</template>
```

Lifecycle hooks

Components lifecycle

Each components instance goes through different states during its life.

The set of states is named **lifecycle**.

Example: *`mounted`*, *`beforeUpdate`*, *`beforeUnmount`*, etc.

Components lifecycle

You can execute JS code when a state is reached:

```
export default {
  mounted() {
    console.log(`the component is now mounted.`)
  }
}
```

Components lifecycle

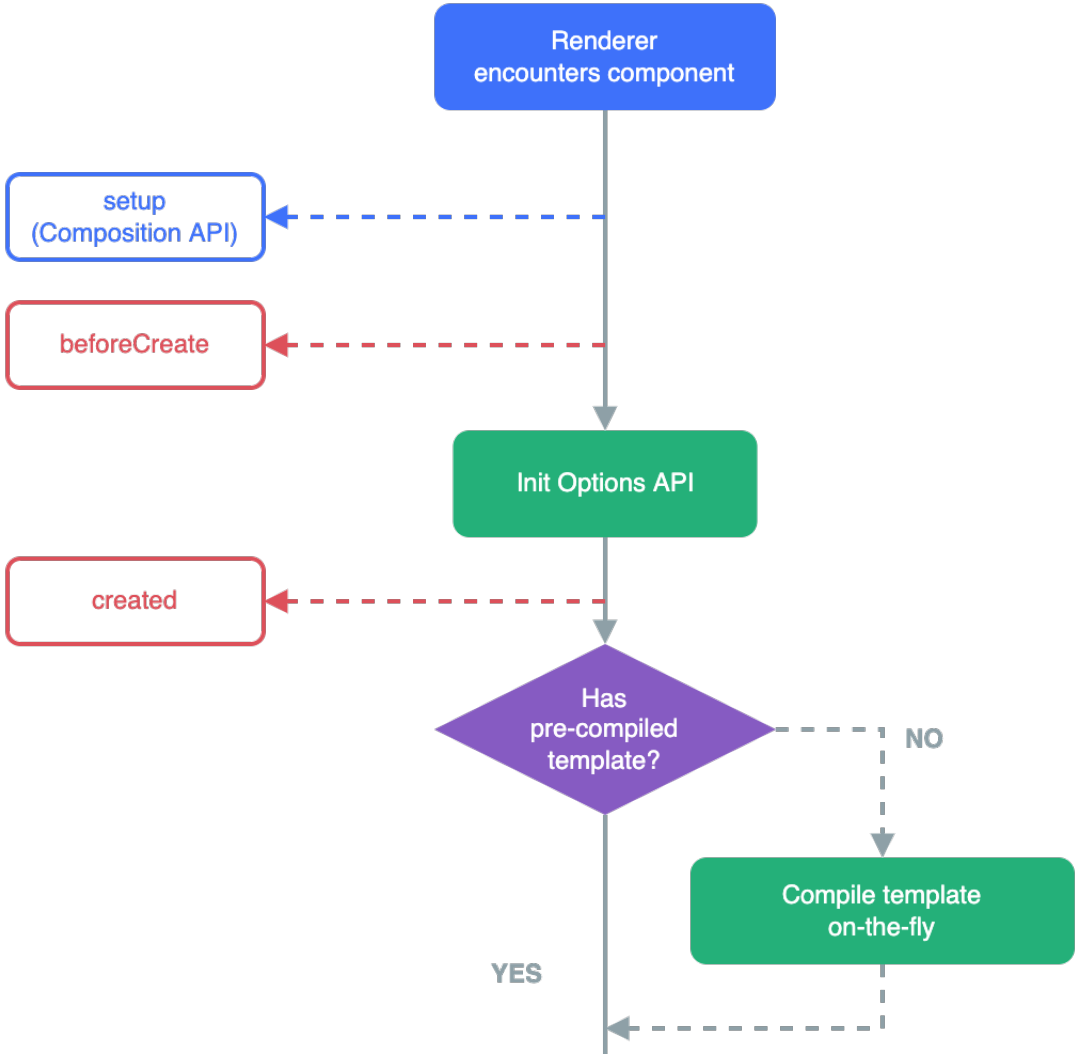

Image (C) by Vue.js documentation

Components lifecycle

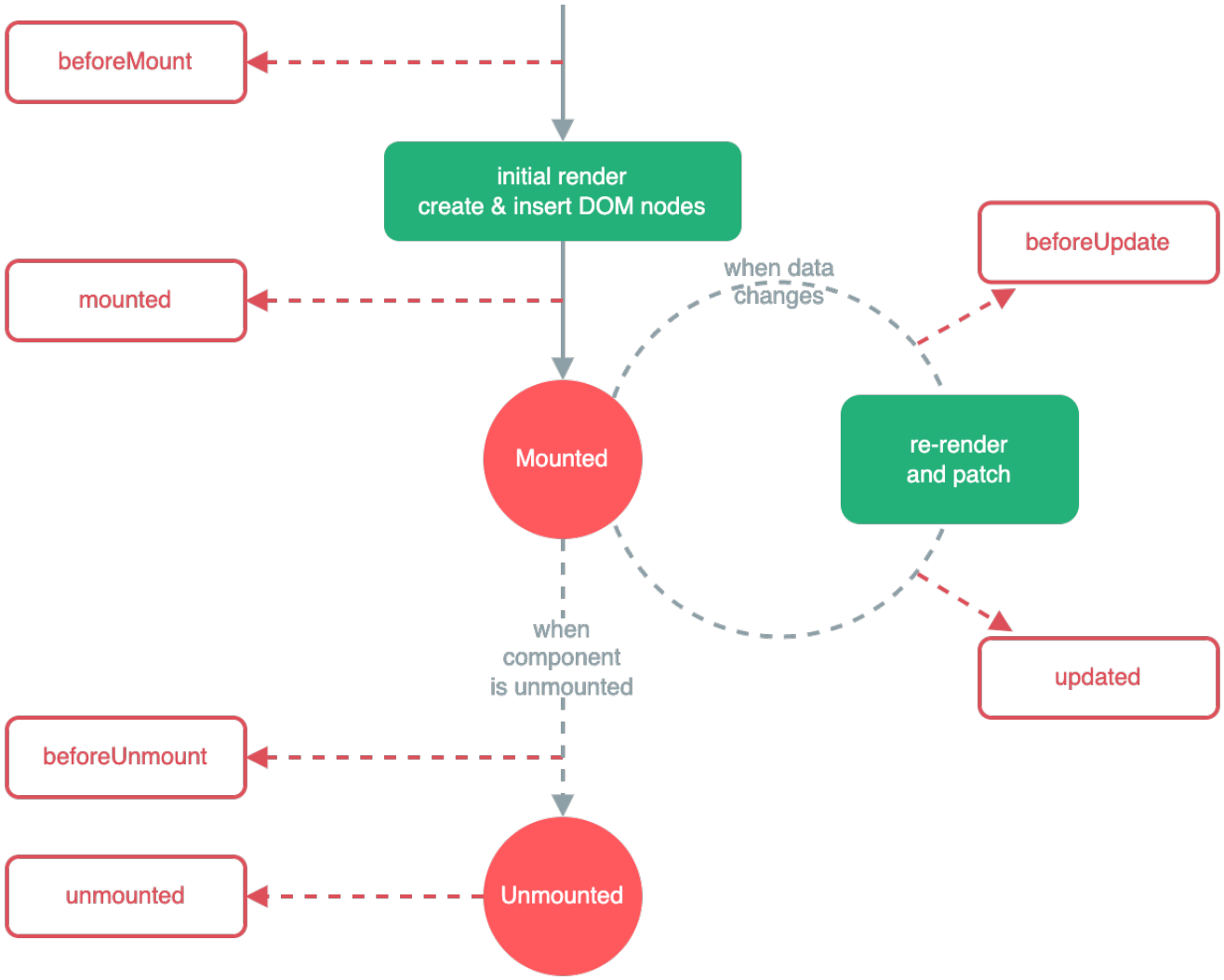

Image (C) by Vue.js documentation

Links

- https://vuejs.org/guide/essentials/lifecycle.html
- https://vuejs.org/api/options-lifecycle.html