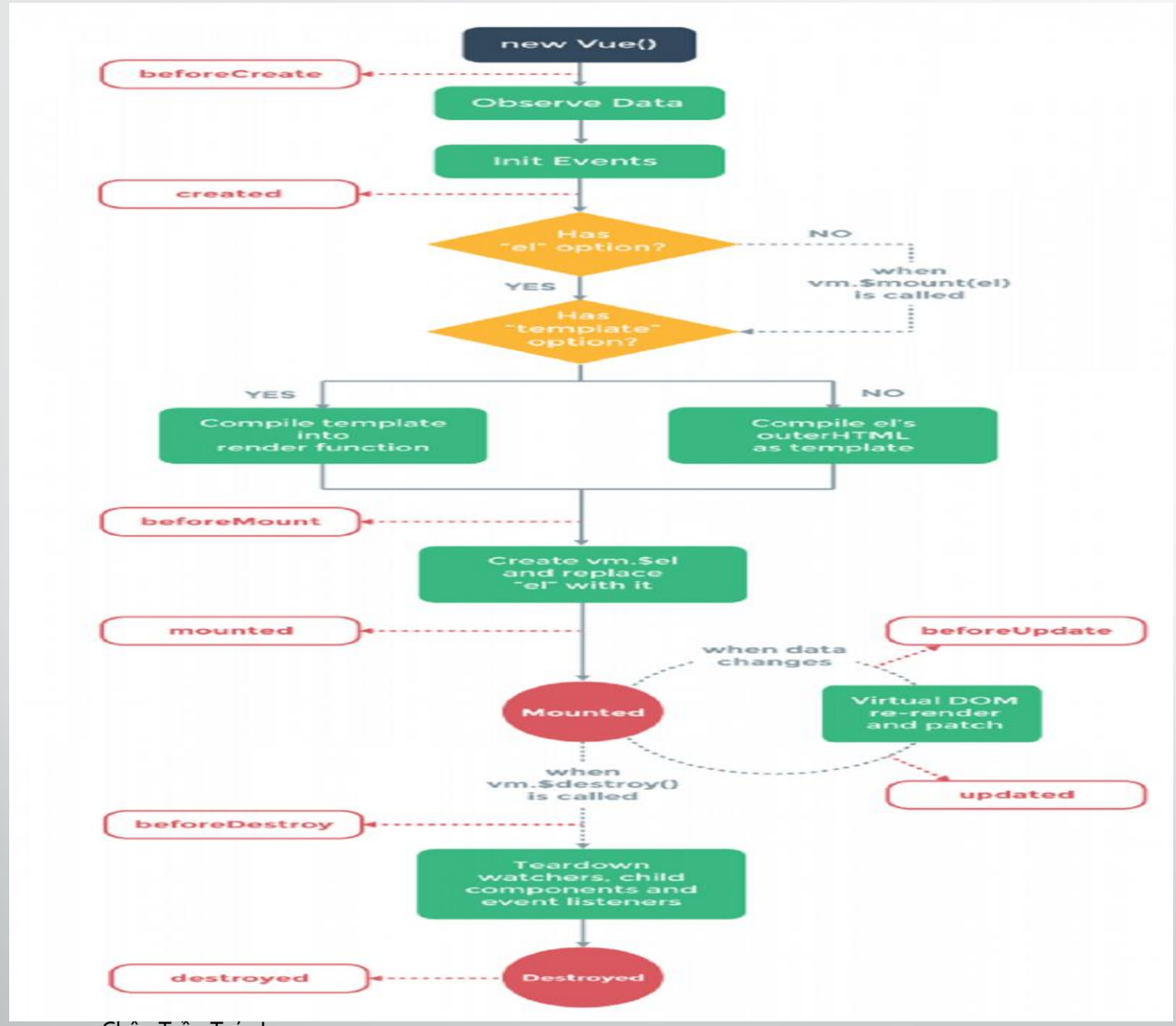
VÒNG ĐỜI CỦA VUE 3X

NỘI DUNG CHÍNH

I SƠ ĐỒ VÒNG ĐỜI

II CÁC GIAI ĐOẠN

1. Sơ đồ vòng đời



1. Sơ đồ vòng đời

Dưới đây là vòng đời của 1 Vue instance

Creating => Mounting => Updating => Destroying

1.Creating: Khởi tạo đối tượng

2.Mounting: Gắn đối tượng với DOM

3.Updating: Cập nhật đối tượng trên DOM

4.Destroying: Huỷ đối tượng

2. Các giai đoạn

2.1 beforeCreate()

Hook beforeCreate sẽ được gọi đồng bộ ngay sau khi Vue được khởi tạo. Các data (dữ liệu) và event (sự kiện) chưa được thiết lập. Ví dụ với đoạn code sau:

```
data() {
  return {
    msg: "Hi, anonymous. Welcome to Yeulaptrinh.vn!"
  }
},
beforeCreate() {
  console.log("-------beforeCreate--------")
  console.log(this.msg)
},
```

2. Các giai đoạn

2.2 Created()

Lúc này, các data và event đã được thiết lập.

Ví dụ với đoạn code sau:

```
data() {
  return {
    msg: "Hi, anonymous. Welcome to Yeulaptrinh.vn!"
  }
},
beforeCreate() {
  console.log("-------beforeCreate--------")
  console.log(this.msg)
},
created() {
  console.log("-------Created--------")
  console.log(this.msg)
},
```

2. Các giai đoạn

2.3 beforeMount

beforeMount được gọi sau khi component đã được compile và trước lần render đầu tiên. Ở giai đoạn này khi truy cập đến các phần tử trong DOM vẫn sẽ báo lỗi:

```
<template>
    <div id="my-text">
        This is my text
    </div>
</template>

<script>
    export default {
        beforeMount() {
            console.log(this)
            console.log(document.getElementById('my-text').innerHTML)
         }
    }
</script>

<style lang="scss" scoped>
</style>
```

2. Các giai đoạn

2.4. Mounted

Trong giai đoạn này, mounted hook sẽ cho phép chúng ta có thể truy cập vào phần tử trong DOM. Tức là khi này, DOM đã được gắn kết.

Mounted là khi mà chúng ta đã nhìn thấy nội dung ở trên trình duyệt.

2. Các giai đoạn

2.4. Mounted

```
<template>
    <div id="my-text">
        This is my text
    </div>
</template>

<script>
    export default {
        mounted() {
            console.log(this.$el)
            console.log(document.getElementById('my-text').innerHTML)
         }
    }
</script>

<style lang="scss" scoped>
</style>
```

2. Các giai đoạn

2.5. Updating (Thay đổi và Re-render)

Quá trình này được gọi bất cứ khi nào các dữ liệu **reactive** bị thay đổi hoặc ta tác động khiến cho **component phải re-render.**

Các hàm sử dụng trong quá trình này thường sử dụng trong debug xem khi nào **component re-render**, còn nếu các bạn muốn debug với các dữ liệu reactive thì nên dùng **computed hoặc watch**

2. Các giai đoạn

2.6. beforeUpdate()

Hook này được gọi ngay sau khi có sự thay đổi dữ liệu (data) trên component. Và được thực hiện trước khi DOM re-render. Bạn có thể lấy được dữ liệu mới (new data) tại đây.

2. Các giai đoạn

2.6. beforeUpdate()

```
<template>
    <div id="my-text">
        {{ counter }}
    </div>
</template>

<script>
    export default {
      data() {
        return {
          counter: 0
        }
      },
      beforeUpdate() {
        console.log(this.counter) // Logs the counter value every second,
      },
      created() {
        setInterval(() => {
          this.counter++
        }, 1000)
      }
}
</script>

<style lang="scss" scoped>
```

2. Các giai đoạn

2.6. beforeUpdate()

```
<template>
    <div  class = "pull-right" >{{ data }}</div>
</template>


<script>
    export default {
        data() {
            return {
                data : 0
            }
        },
        created() {
            setInterval(() => {
                this .data++
                console.log( "Change data count of component: " +  this .data)
            }, 1000)
        },
        beforeUpdate() {
            console.log( "Data beforeUpdate: " +  this .data)
        }
    }
</script>
```

2. Các giai đoạn

2.7. Update()

Sau hook beforeUpdate(), sau khi DOM đã re-render. Dữ liệu truy xuất được là dữ liệu sau khi được thay đổi của component, cũng là dữ liệu lấy được trong beforeUpdate().

Một chú ý ở đây: Nên tránh việc thay đổi data khi sử dụng hook này. Mà thay vào đó hãy sử dụng các thuộc tính watcher hay computed.

2. Các giai đoạn

2.7. Update()

```
<template>
    <div class = "pull-right" >{{ data }}</div>
</template>


<script>
    export default {
        data() {
            return {
                data : 0
            }
        },
        created() {
            setInterval(() => {
                this.data++
                console.log( "Change data count of component: " + this.data)
            }, 1000)
        },
        updated() {
            console.log( "Data updated: " + this.data)
        },
        beforeUpdate() {
            console.log( "Data beforeUpdate: " + this.data)
        }
    }
</script>
```

2.8. Destruction (Huỷ bỏ)

Destruction Hook dùng để thực hiện các hành động khi component của bạn bị huỷ bỏ. Hay nói cách khác là xoá khỏi DOM. Nếu bạn sử đã từng sử dụng Vue Router hay tạo các ứng dụng SPA thì chắc chắn sẽ hiểu rõ. Việc huỷ bỏ component cũ và thay thế component mới sẽ tiết kiệm bộ nhớ rất nhiều cũng như cải thiện tốc độ. Đó cũng là một điểm mạnh của Vue component.

2. Các giai đoạn

2.8. Destruction (Huỷ bỏ)

beforeDestroy()

beforeDestroy() hook được gọi ngay trước khi huỷ bỏ component. Đây là giai đoạn thích hợp nhất để bạn xoá bỏ các data, events để dọn dẹp.

destroyed()

destroyed() hook được gọi khi component đã bị xoá bỏ khỏi DOM.

2. Các giai đoạn

2.8. Destruction (Huỷ bỏ)

```
<template>
    <div id= "root" class = "pull-right" >
      <test-component v- if = "show" ></test-component>
      <button  class = "btn btn-default" @click= "show = !show" >Action</button>

    </div>
</template>

<script>
    import Vue from  'vue'

    Vue.component( 'test-component' , {
        template:  '<div>VueJS {{ content }}</div>' ,
        data() {
            return {
                content :  'ITMagical' ,
                interval :  ''
            }
        },
```

2. Các giai đoạn

2.8. Destruction (Huỷ bỏ)

```
        beforeDestroy() {
            console.log( 'beforeDestroy' )
        },
        destroyed() {
            console.log( 'destroyed && content = ' + this .content)
        },
        created() {
            this .interval = setInterval(() => {
                console.log( 'not removed' )
            },1000)
        }
    })

    export default {
        data() {
            return {
                show :  true
            }
        }
    }
</script>
```

2. Các giai đoạn

2.8. Destruction (Huỷ bỏ)

```
<template>
    <div id= "root" class = "pull-right" >
      <test-component v- if = "show" ></test-component>
      <button  class = "btn btn-default" @click= "show = !show" >Action</button>

    </div>
</template>

<script>
    import Vue from 'vue'

    Vue.component( 'test-component' , {
        template:  '<div>VueJS {{ content }}</div>' ,
        data() {
            return {
                content :  'ITMagical' ,
                interval :  ''
            }
        },
```

2. Các giai đoạn

2.8. Destruction (Huỷ bỏ)

```
        beforeDestroy() {
            this.content = null
            delete this.content
            clearInterval( this.interval)
            console.log( 'beforeDestroy' )
        },
        destroyed() {
            console.log( 'destroyed && content = ' + this.content)
        },
        created() {
            this.interval = setInterval(() => {
                console.log( 'not removed' )
            },1000)
        }
    })


    export default {
        data() {
            return {
                show : true
            }
        }
    }
</script>
```

2. Các giai đoạn

Ở Vue 3 thì beforeDestroy và destroyed được đổi tên thành beforeUnmount và unmounted.

Vue 3 cũng cung cấp cho chúng ta các event mới để ta có thể bắt được các mount, unmount, updated,... để tiện hơn cho chúng ta nữa, công dụng thì cũng giống như các hooks mounted/unmounted,...


the
end