VUE3 新特性

Vue3 目前已经发布 RC1，距离最终正式发布应该不远了，从内容来说应该新特性也都已经确定了，目前入手应该算是不错的时间点。 从已经了解的特性来说，大概罗列如下

语言特性

- 重构数据响应实现，从 Object.defineProperty 到 Proxy，可以对新增属性进行响应了
- Typescript

快速小巧，性能优化

- 虚拟 DOM 重构，性能优化
- Tree Shaking，就是按需引入，自动修剪

规模性

- Typescript
- Composition API，也可以称为 Vue Hook，相比来说应用中最大的变化

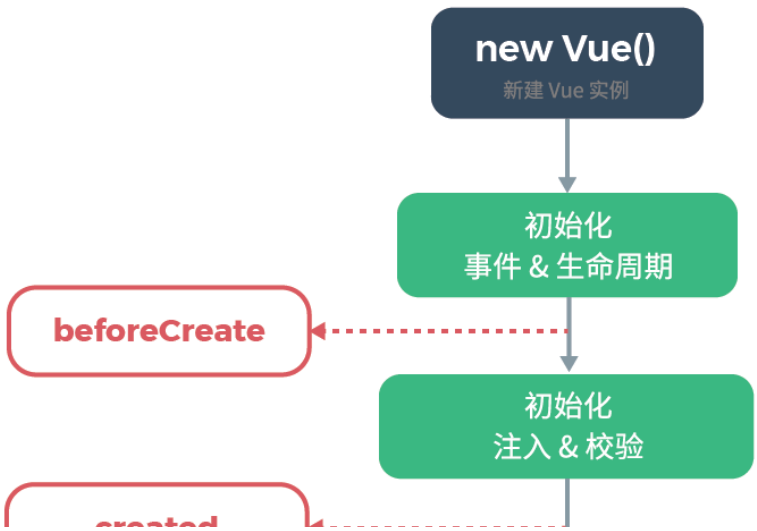
其它

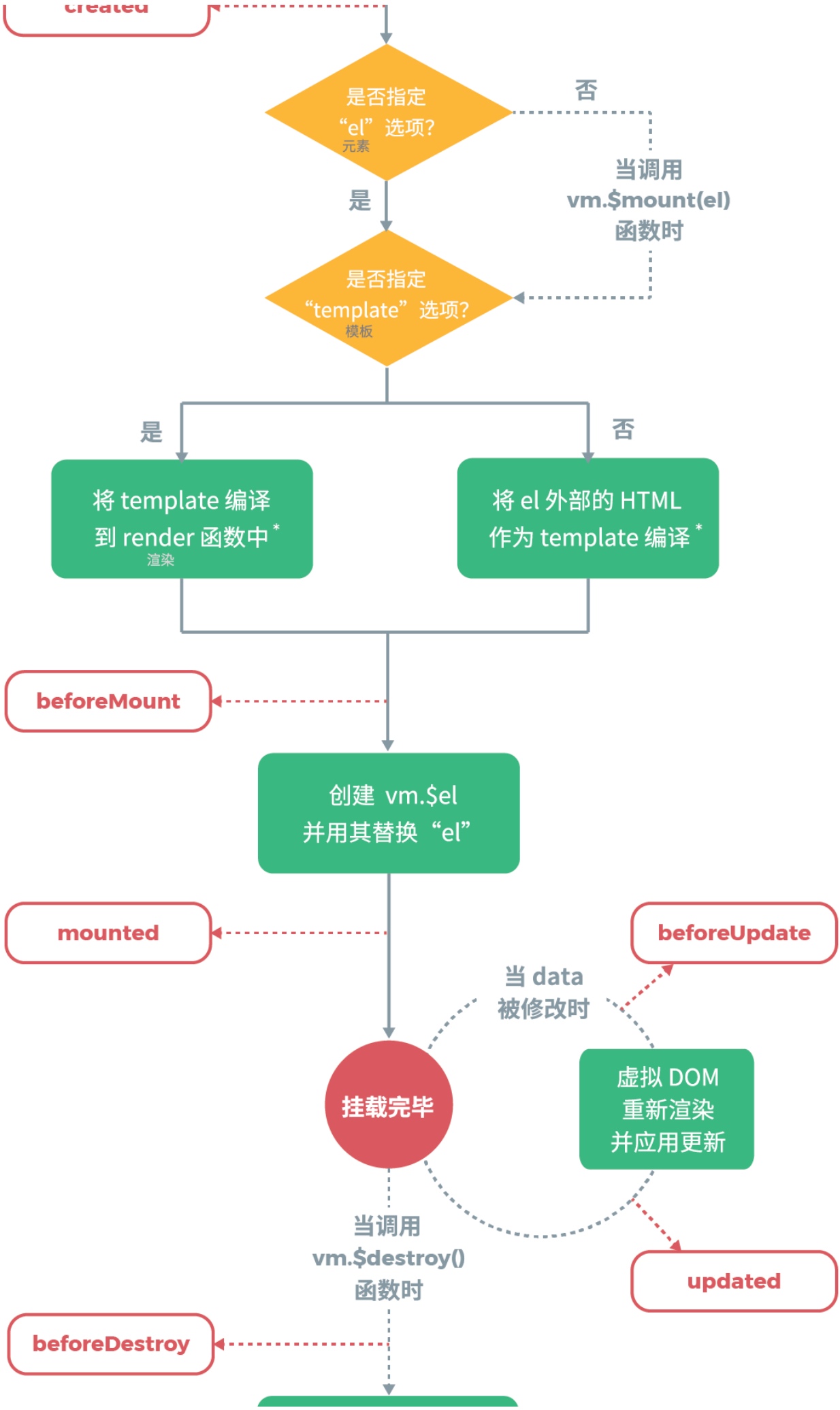
- 取消 Vue 全局变量
- 自定义 Directive API 的调整
- 自定义组件支持 v-model
- Fragment Template 支持多个根结点
- Suspense
- Teleport

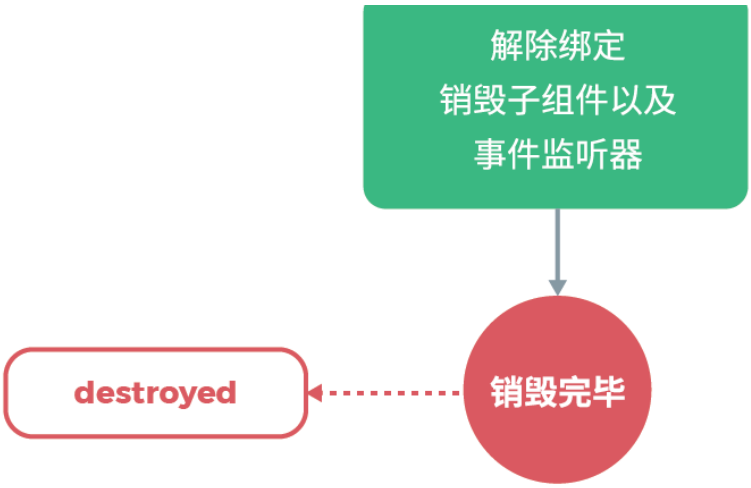
从新版本的特性选择和原因解释中，Class Component 和 React 的发展类似，处于半放弃的状态，Hook 方式都表明 Function-based 方向的确认。

Vue3 生命周期

下面是 Vue2 的生命周期图:




created
是否指定
“el”选项?
元素
否
当调用
vm.$mount(el)
函数时
是
是否指定
“template”选项?
模板
是
否
将 template 编译
到 render 函数中*
渲染
将 el 外部的 HTML
作为 template 编译*
beforeMount
创建 vm.$el
并用其替换“el”
mounted
beforeUpdate
当 data
被修改时
挂载完毕
虚拟 DOM
重新渲染
并应用更新
当调用
vm.$destroy()
函数时
updated
beforeDestroy



* 如果使用构造生成文件（例如构造单文件组件），
模板编译将提前执行

在 Vue3 中，变化如下

替换

- beforeCreate -> setup()
- created -> setup()

重命名

- beforeMount -> onBeforeMount
- mounted -> onMounted
- beforeUpdate -> onBeforeUpdate
- updated -> onUpdated
- beforeDestroy -> onBeforeUnmounted
- destroyed -> onUnmounted
- errorCaptured -> onErrorCaptured

新增 新增的两个主要用于方便 debug 回调

- onRenderTracked
- onRenderTriggered

在 Vue3.x 中，因为兼容 2.x 的语法，所有旧的生命周期函数得到保留（除了已经更名删除的），当生命周期混合使用时，3.x 生命周期相对优先 2.x 的执行，比如 onMounted 比 mounted 先执行。

Composition API 以补丁方式引入到 2.x 中，在 2.x 中混用时，和 3.x 相反，2.x 的回调会优先执行。

这两种情况比较下会发现，用哪个版本，哪个版本的特性会优先。因此两个版本下的执行因为优先相反，有可能造成不同的结果。因此最好的建议是不要混用。

Composition API

这个部分应该是 Vue3 当中对于使用者最大的新增特性，单独说明。

取消 Vue 全局变量

基于 Function-based 模式，2.x 中的全局变量 Vue 被取消，改为实例函数 createApp()创建实例对象。下面是 3.x 中创建实例对象的方式。

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

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

对比可以看一下 2.x 的方式

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

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
    el: '#app',
    router,
    template: '<App/>',
    components: { App }
})
```

从使用角度，这里对使用者的影响不大，CLI 部分会生成对应的代码，除非自己创建框架代码的时候才有更多了解的需要。

自定义指令 Directives API

这里的变化主要是因为前面提及的生命周期调整，因此自定义指令中对应调整

2.x 支持的钩子函数

```
const MyDirective = {
    bind(el, binding, vnode, prevVnode) {},
    inserted() {},
    update() {},
    componentUpdated() {},
    unbind() {}
}
```

3.x 修改如下

```
const MyDirective = {
    beforeMount(el, binding, vnode, prevVnode) {},
    mounted() {},
    beforeUpdate() {},
    updated() {},
    beforeUnmount() {}, // new
    unmounted() {}
}
```

Component 组件支持 v-model 指令

定制组件现在也可以使用 v-model 来绑定属性，2.x 中标准 HTML 可以像下面这样使用 v-model 绑定：

```
<input v-model="searchText" />
```

这样的定义和下面的定义是等同的:

```
<input :value="searchText" @input="searchText = $event.target.value" />
```

在 3.x 当中，v-model 可以用在自定义组件中：

```
<custome-input v-model="searchText" />
```

它的作用和下面的定义一致：

```
<custom-input
    :model-value="searchText" @update:model-value="searchText = $event">
</custom-input>
```

如果需要绑定多个，可以如下设置：

```
<text-document
    v-model:title="doc.title"
    v-model:content="doc.content"
></text-document>
```

Fragments Template 支持多个根结点

2.x 的 Template 模版中只能放置一个根节点，比如

```
<template>
    <div>
        <p>Hello</p>
        <p>Vue2.x</p>
    </div>
</template>
```

假如你有需要放置两个根结点，那么一个是你再建立一个 Vue 实例，或者放置一个根结点将两个或者多个节点放进去，在 3.x 中，可以不用唯一根结点，变的简单了

```
<template>
    <h1>{{ msg }}</h1>
    <button @click="handleClick">count is: {{ count }}</button>
    <p>
        Edit <code>components/HelloWorld.vue</code> to test hot module
        replacement.
    </p>
</template>
```

Suspense Template Fallback 组件

Suspense 组件主要用在异步组件处理中用于显示不同的状态。

Async components are suspensible by default. This means if it has a in the parent chain, it will be treated as an async dependency of that . In this case, the loading state will be controlled by the , and the component's own loading, error, delay and timeout options will be ignored.

The async component can opt-out of Suspense control and let the component always control its own loading state by specifying suspensible: false in its options.

Teleport Template Dom 占位传递组件

Vue 和很多前端框架都鼓励我们能够创建封装好的组件，组件能够尽量独立，组件自身的逻辑组织在一起，和其它部分相对独立。但是因为前端显示时一些技术限制的要求，有时需要将组件中的一些部分移动到 DOM 树的其它位置，从而造成组件的边界封装以及独立的要求无法控制。例如一个模式对话框的显示

```
<template>
    <div class="comp-class">
        <button @click="modalOpen = true">打开对话框</button>

        <!-- 对话框部分 -->
        <div v-if="modalOpen" class="modal">
            <p>这是一个对话框！</p>
            <button @click="doSomething">确定</button>
```

```
            <button @click="modalOpen = false">取消</button>
        </div>

    </div>
</template>
```

上面的代码当我们做一个弹窗的时候，按照业务逻辑，弹窗和其它代码在一起。但是这样写往往会出现问题，就是我们点击“删除用户”，会显示弹窗，让用户确认/取消，但是这个弹窗可能会被外面的元素挡住，这通常是`z-index`的原因。

为了能够正常显示，我们就会想办法把这个弹出框直接挂在`body`节点上，这样做显示上没有问题，但是这样做的话：**业务逻辑和代码连贯性都出问题了**。

为了解决这个烦恼，3.x 新增了组件来解决这个问题，我们首先需要在最外层预留一个显示块

```
<!-- 预留一块空地，专门用来显示这个容易被遮挡的层 -->
<div id="modal-container"></div>

<!-- app -->
<div id="app">
```

组件当中这样写：

```
<template>
    <div class="comp-class">
        <button @click="modalOpen = true">打开对话框</button>

        <!-- 对话框部分 -->
        <Teleport to="#modal-container">
            <div v-if="modalOpen">
                <p>这是一个对话框</p>
                <button @click="doSomething">确定</button>
                <button @click="modalOpen = false">取消</button>
            </div>
        </Teleport>

    </div>
</template>
```

这样代码的完整性，业务连贯性都得到了保障，弹窗的显示问题也可以解决了。