## vue生命周期（钩子函数）

### 问题

请说一下vue的生命周期函数（钩子函数）。

### 描述

首先关于生命周期函数，一般我的第一个问题就是这个，我认为是每个使用vue的都要清楚的，如果这个问题答的问题很大其实我都不太想继续往下进行了。  
即使英语不标准（我就是不标准的人，并不是说这是个问题）也要去把关键点说清楚，哪个地方有ed哪个地方没有ed其实是很关键的，或者可以手写下来，因为常用的就是created和mounted所以前4个可以清楚的手写出来并不难，后面4个不去详细说明都没事。（我自己工作中基本没用过后面4个）  
在哪个周期能够首次拿到data数据和在哪个周期能够首次拿到mounted中的dom元素，如果没有说到这个问题，我一般会一直往下问，直到他说出来这两个答案。

### 答案

beforeCreate、created（此时需说明可以在created中首次拿到data中定义的数据）、beforeMount、mounted（此时需说明dom树渲染结束，可访问dom结构）、beforeUpdate、updated、beforeDestroy、destroyed

## computed中的getter和setter

### 问题

说一下computed中的getter和setter。

### 问题描述

很多情况，我问到这个问题的时候对方的回答都是vue的getter和setter、订阅者模式之类的回答，我就会直接说问的并不是这个，而是computed，直接让对方说computed平时怎么使用，很多时候得到的回答是computed的默认方式，只使用了其中的getter，就会继续追问如果想要再把这个值设置回去要怎么做，当然一般会让问到这个程度的这个问题他都答不上来了。

### 答案

<!--直接复制的官网示例-->

computed: {

fullName: {

// getter

get: function () {

return this.firstName + ' ' + this.lastName

},

// setter

set: function (newValue) {

var names = newValue.split(' ')

this.firstName = names[0]

this.lastName = names[names.length - 1]

}

}

}

## watch监听对象

### 问题：如何watch监听一个对象内部的变化。

### 描述：这个问题我感觉是一个不应该不会的问题，可是我遇到的人大部分都没有给出我所期望的答案，有些人会说直接监听obj，好一点的会说直接点出来监听obj.key，但是很少有人回答deep，开始我还会去问immediate，但是太多人不知道了，所以后来我就只问监听对象了，只有回答出deep的才会去问immediate的作用。

### 答案

如果只是监听obj内的某一个属性变化，可以直接obj.key进行监听。

watch: {

'obj.question': function (newQuestion, oldQuestion) {

this.answer = 'Waiting for you to stop typing...'

this.debouncedGetAnswer()

}

}

如果对整个obj深层监听

watch: {

obj: {

handler: function (newQuestion, oldQuestion) {

this.answer = 'Waiting for you to stop typing...'

this.debouncedGetAnswer()

},

deep: true,

immediate: true

}

}

immediate的作用：当值第一次进行绑定的时候并不会触发watch监听，使用immediate则可以在最初绑定的时候执行。

## v-for循环key的作用

### 问题：v-for循环时为什么要加key。

### 描述：问这个问题时，好多人再先回答的都是页面有警告，编辑器有提示，我会直接说不考虑报错和提示的问题，或者会问如果不加key的话，页面会不会出现什么异常情况。有的人会说是一个标识，标识他的唯一性，我会继续追问为什么要标识唯一性呢，不加又怎么样？

### 答案

vue的dom渲染是虚拟dom，数据发生变化时，diff算法会只比较更改的部分，如果数据项的顺序被改变，Vue将不是移动DOM元素来匹配数据项的改变，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。举例说明：有一个列表我们现在在中间插入了一个元素，diff算法会默认复用之前的列表并在最后追加一个，如果列表存在选中一类的状态则会随着复用出现绑定错误的情况而不是跟着原元素，key的作用就可以给他一个标识，让状态跟着数据渲染。（这一块是我自己的一个大概理解，表述不太清楚，具体的可以去查一下文档，本文就不具体描述此问题了。）

## $nextTick

### 问题：$nextTick用过吗，有什么作用。

### 描述：问到这个问题时，很多人都会说到可以处理异步，而往下追问为什么要用nextTick，他解决了什么问题，不用他会怎么样的时候就很多人说不上来了。

### 答案

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。(官网解释)  
解决的问题：有些时候在改变数据后立即要对dom进行操作，此时获取到的dom仍是获取到的是数据刷新前的dom，无法满足需要，这个时候就用到了$nextTick。

// 修改数据

vm.msg = 'Hello'

// DOM 还没有更新

Vue.nextTick(function () {

// DOM 更新了

})

// 作为一个 Promise 使用 (2.1.0 起新增，详见接下来的提示)

Vue.nextTick()

.then(function () {

// DOM 更新了

})

## $set

### 问题：vue中的$set用过吗，为什么要用它，解决了什么问题

### 描述：这个问题知道的人就基本都能说出来，但是不知道的就是一点不了解，有的还会说到es6的set结构

### 答案

向响应式对象中添加一个属性，并确保这个新属性同样是响应式的，且触发视图更新。它必须用于向响应式对象上添加新属性，因为 Vue 无法探测普通的新增属性 (比如 this.myObject.newProperty = 'hi')（官方示例）

我自己的理解就是，在vue中对一个对象内部进行一些修改时，vue没有监听到变化无法触发视图的更新，此时来使用$set来触发更新，使视图更新为最新的数据。

## 组件间的传值

### 问题：说一下组件间的传值方式，你知道的所有方式都说一下

### 描述：这个问题其实就是想看官方文档有没有具体看过，因为很多传值方式官方文档上有描述，但是项目中用的相对较少。

基本都能回答上来，父传子：props；子传父：$emit；兄弟：eventbus；vuex；有一些会说到sessionStorage和localStorage、路由传参（这个答案其实并不是我想要问的，不过也可以实现一定的传值）

以下传值方式的具体使用方式本文不具体描述了，后面会再写一篇关于传值方式的文章，到时候再附上链接。

### 答案

1. provide / inject  
   这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。
2. Vue.observable  
   让一个对象可响应。Vue 内部会用它来处理 data 函数返回的对象。  
   返回的对象可以直接用于渲染函数和计算属性内，并且会在发生改变时触发相应的更新。也可以作为最小化的跨组件状态存储器，用于简单的场景：

const state = Vue.observable({ count: 0 })

const Demo = {

render(h) {

return h('button', {

on: { click: () => { state.count++ }}

}, `count is: ${state.count}`)

}

}

1. $attrs  
   包含了父作用域中不作为 prop 被识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="$attrs" 传入内部组件——在创建高级别的组件时非常有用。
2. $listeners  
   包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件——在创建更高层次的组件时非常有用。
3. props
4. $emit
5. eventbus
6. vuex

7. $parent / $children / ref

内容：

<slot>元素作为承载分发内容的出口

合成组件(插槽)

<navigation-link url="/profile>

Profile

</navigation-link>

使用

<a v-bind:href="url" class="nav-link" >

<slot></slot>

<a>

emit

父组件可以使用props把数据传给子组件

子组件可以使用$emit触发父组件的自定义事件

被import的是子组件

vm.$emit( event, arg) //触发当前实例上对象

vm.$on( event, fn); //监听event事件后运行fn