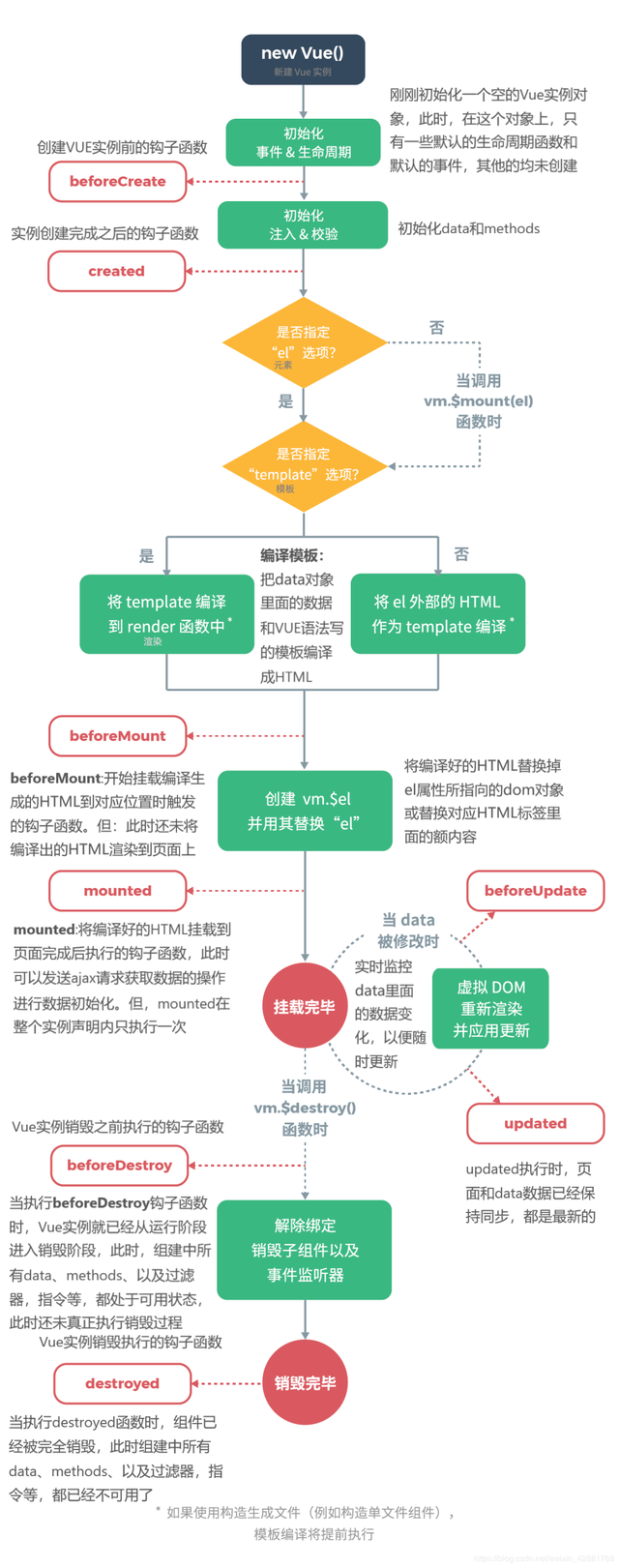
随笔- 543  文章- 2  评论- 4

[**vuejs生命周期函数**](https://www.cnblogs.com/wzndkj/p/9612647.html)



**生命周期函数就是vue实例在某一个时间点会自动执行的函数**

当我们创建一个实例的时候，也就是我们调用 new Vue() 这句话的时候，vue会帮助我们去创建一个实例，创建过程其实并不像我们想的那么简单，他要经过很多的步骤

Init(Events & Lifecycle):首先他会去初始化事件和生命周期相关的内容,当最基础的初始化完成的时候，在这个时间点上，vue会自动的帮我去之行一个函数，这个函数就是beforeCreate

**BeforeCreate**

该函数执行在组件创建、数据观测 (data observer) 和 event/watcher 事件配置之前，实例初始化之后被调用。

在该阶段组件未创建，不能访问数据，组件中的 data，ref 均为 undefined。

**Created**

该函数在组件创建完成后被立即调用，在这一步，实例已完成以下的配置：数据观测 (data observer)，属性和方法的运算，watch/event 事件回调。

但是还未渲染成HTML模板，组件中的data对象已经存在，可以对data进行操作了，即可以访问数据，发请求，ref依旧是undefined，挂载阶段还没开始，$el 属性目前尚不可用。

一般我们可以将对数据的初始化和初始化页面的请求放到里面，结束loading。

**BeforeMount**

该函数在组件挂载之前，在该阶段页面上还没渲染出 HTML 元素，data 初始化完成，ref 依旧不可以操作，相关的 render 函数首次被调用。

可以访问数据，编译模板结束，虚拟 dom 已经存在。

该钩子在服务器端渲染期间不被调用。

模板结合数据会被挂载到页面上，dom挂载到页面之上

**Mounted**

该函数是页面完成挂载之后执行的，这时 el 被新创建的 vm.$el 替换了，就可以操作 ref 了，一般会用于将组件初始时请求数据的方法放到这里面，filter 也是在这里生效。

如果根实例挂载到了一个文档内的元素上，当 mounted 被调用时 vm.$el 也在文档内。

可以拿到数据和节点，实例被挂载后调用。

注意 ：mounted 不会保证所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以在 mounted 内部使用 vm.$nextTick：

mounted: function () {this.$nextTick(function () {// Code that will run only after the// entire view has been rendered })}

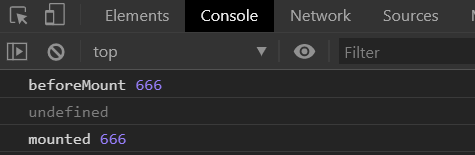
该钩子在服务器端渲染期间不被调用。

**1、beforeMount与mounted实例①：**

（ref表示节点；this.$refs.ref的标识，就可拿到节点）

<divid="box"><divref='demo'>demo</div></div><script>new Vue({el:"#box",data:{a:666 }, mounted(){console.log('mounted',this.a); }, beforeMount(){console.log('beforeMount',this.a);console.log(this.$refs.demo); } })</script>

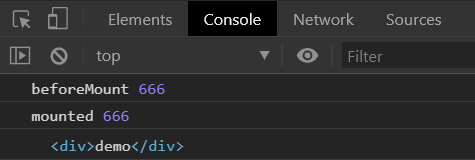
**结果：**



**2、beforeMount与mounted实例②：**

<divid="box"><divref='demo'>demo</div></div><script>new Vue({el:"#box",data:{a:666 }, mounted(){console.log('mounted',this.a);console.log(this.$refs.demo); }, beforeMount(){console.log('beforeMount',this.a); } })</script>

**结果：**

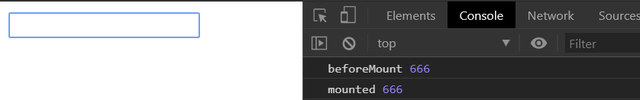


**3、beforeMount与mounted实例③：**

（节点挂载后，文本框自动获取焦点）

<divid="box"><inputtype="text"ref='txt'></div><script>new Vue({el:"#box",data:{a:666, }, mounted(){console.log('mounted',this.a);this.$refs.txt.focus(); }, beforeMount(){console.log('beforeMount',this.a); } })</script>

**结果：**



以上两个实例，更加有力的说明：

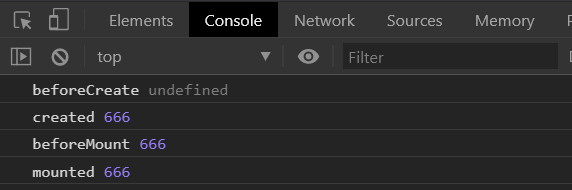
beforeMount执行在真实的dom节点挂载之前，此时没有节点，所以拿不到节点。mounted执行时真实的dom节点已经挂载到页面上了，所以能拿到节点。

例：beforeCreate、created、beforeMount、mounted执行顺序差别演示：

（因为 它们此时都是对象，所以它们代码的摆放位置是不影响结果的。）

<divid="box"></div><script>new Vue({el:"#box",data:{a:666, }, created(){console.log("created",this.a); }, beforeCreate(){console.log('beforeCreate',this.a); }, mounted(){console.log('mounted',this.a); }, beforeMount(){console.log('beforeMount',this.a); } })</script>

**结果：**



**beforeUpdate,updated**:

[复制代码](javascript:void(0);)

var vm = new Vue({

　　el:'#app',

　　template:'<h1>hello</h1>',

　　beforeUpdate:function(){

　　　　console.log('before updated')

　　},

　　updated:function(){

　　　　console.log('updated')

　　}

})

[复制代码](javascript:void(0);)

刷新页面看，发现这两个钩子函数其实并没有被执行，那为什么没有被执行呢，看图解释说是，when data changes，当数据发生改变的时候才会被执行

beforeUpdate：数据发生改变，还没有被渲染之前，beforeUpdate会被执行

beforeUpdate

只要是页面数据改变了都会触发，数据更新之前，页面数据还是原来的数据，当你请求赋值一个数据的时候会执行这个周期，如果没有数据改变不执行。

该函数在数据更新时调用，发生在虚拟 DOM 打补丁之前，在有特殊需求的情况下，可以将更新之前的数据存起来，放到后面去使用。

这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。

该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务端进行。

updated

只要是页面数据改变了都会触发，数据更新完毕，页面的数据是更新完成的。beforeUpdate和updated要谨慎使用，因为页面更新数据的时候都会触发，在这里操作数据很影响性能和容易死循环。

不要在updated里面修改属性，修改属性会进入死循环；【推荐使用update 因为watch太消耗资源】

updated:当数据重新渲染之后，updated这个生命周期函数会被执行

由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子，在数据更新之后做一些处理，即监控数据的变化。

当这个钩子被调用时，组件 DOM 已经更新，所以你现在可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态。如果要相应状态改变，通常最好使用计算属性或 watcher 取而代之。

注意 ：updated 不会保证所有的子组件也都一起被重绘。如果你希望等到整个视图都重绘完毕，可以在 updated 里使用 vm.$nextTick：

updated: function () {this.$nextTick(function () {// Code that will run only after the// entire view has been re-rendered })}

该钩子在服务器端渲染期间不被调用。

**注意：**

watch是监控特定数据的变化，而updated是监控组件里所有数据的变化。

教程里面只有8个生命周期函数，实际上有11个生命周期函数

**beforeDestroy,destroyed**:

[复制代码](javascript:void(0);)

var vm = new Vue({

　　el:'#app',

　　template:'<h1>hello</h1>',

　　beforeDestroy:function(){

　　　　console.log('beforeDestroy')

　　},

　　destroyed:function(){

　　　　console.log('destroyed')

　　}

})

[复制代码](javascript:void(0);)

刷新页面完毕，这个时候会发现beforeDestroy,destroyed并没有被触发，那什么时候被触发呢

when vm.$destroy() is called:当destroy()这个方法被调用的时候会调用beforeDestroy，当全部销毁的时候，destroyed会被执行，那怎么让他执行呢，在控制台执行vm.$destroy()的时候会调用这两个函数，还没被销毁之前会调用beforeDestroy，已经被销毁后会调用destroyed这个函数

**BeforeDestroy**

这个周期是在组件销毁之前执行，有点类似路由钩子beforeRouterLeave,都是在路由离开的时候执行，只不过beforeDestroy无法阻止路由跳转，但是可以做一些路由离开的时候操作，因为这个周期里面还可以使用data和method。比如一个倒计时组件，如果在路由跳转的时候没有清除，这个定时器还是在的，这时候就可以在这个里面清除计时器。

该函数在实例销毁之前调用，这里的 ref 依旧可以操作，实例仍然完全可用，可以在这里做清除定时器的操作，防止内存泄漏。

该钩子在服务器端渲染期间不被调用。

**Destroyed**

该函数在组件销毁的时候执行，即实例销毁后调用，这里的 ref 不存在。

该钩子被调用后，对应 Vue 实例的所有指令都被解绑，所有的事件监听器被移除，所有的子实例也都被销毁。

该钩子在服务器端渲染期间不被调用。

注：组件进行销毁的时候,是先销毁的是父组件,然后销毁子组件。