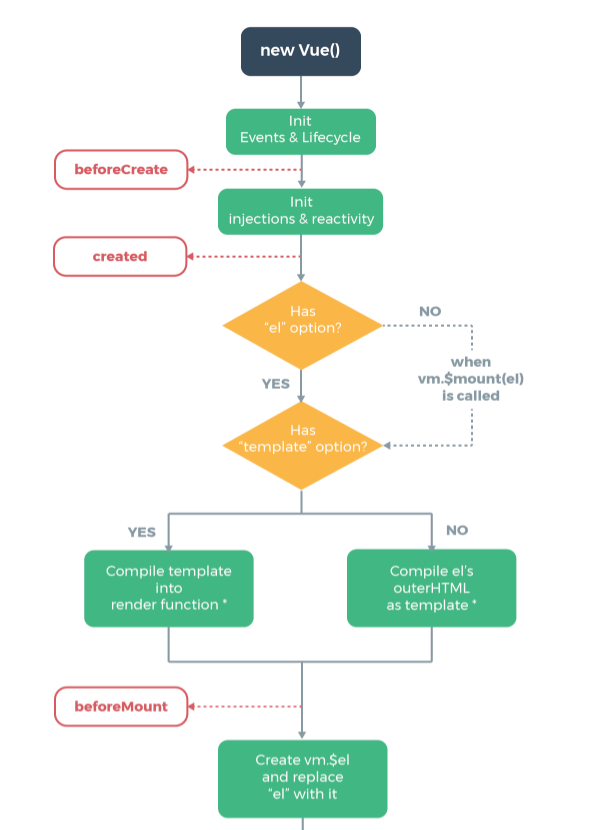
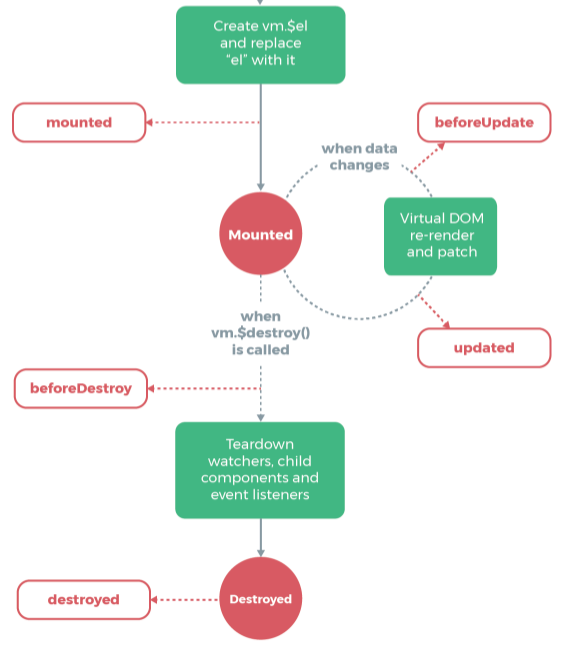
Vue 的生命周期函数：





**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：**既然beforeCreate被自动之行，那么beforeCreate就是一个生命周期函数

代码：

var vm = new Vue({

　　el:'#app',

　　beforeCreate:function(){

　　　　console.log('before create')

　　}

})

我们发现这个在控制台被自动输出了，就是vue自动执行了beforeCreate这个函数，处理完这个函数，vue会继续调用一个写外部的注入，包括双向绑定的相关内容

Init(injections & reactivity): 外部的注射，各种绑定的初始化，这部分初始化完成的时候，基本上vue实例的初始化操作都完成了，在这个结点上，又会有一个自动的函数被执行,这个函数的名字叫created

**created：**这也是一个生命周期函数，因为他完全符合生命周期函数的定义

var vm = new Vue({

　　el:'#app',

　　beforeCreate:function(){

　　　　console.log('before create')

　　},

　　created:function(){

　　　　console.log('created')

　　}

})

可以看到beforeCreate执行之后，created也被自动的执行了，继续看这张图

Has 'el' options:是否有el这个选项

Has 'template' optioins: 是否有template这个属性

　　no->Compile el's outerHtml as template: 如果实例里面没有tempalte这个属性，会把外部el挂载点的html当作模板

　　yes->Compile template into render functoin: 如果实例里面有tempalte，这个时候就会用template去渲染

但是有了模板之后并没有直接渲染到页面上，在渲染之前，又自动到去执行了一个函数，这个函数是beforeMount

**beforeMount**:这个函数也是一个生命周期函数，也就是模板即将挂载到页面到一瞬间，beforeMount会被执行

var vm = new Vue({

　　el:'#app',

　　template:'<h1>hello</h1>',

　　beforeCreate:function(){

　　　　console.log('before create')

　　},

　　created:function(){

　　　　console.log('created')

　　},

　　beforeMount:function(){

　　　　console.log('before mount')

　　}

})

可以看到beforeMount被执行了，在beforeMount执行完成后

Create vm.$el and replace 'el' width it: 模板结合数据会被挂载到页面上，当dom挂载到页面之上，这个时候又有一个生命周期函数被执行了

**mounted**:在beforeMount dom并没有渲染到页面上，在mounted dom已经被渲染到页面上了,这个时候可以做个实验

<div id='app'>

　　hello world　　</div>

<script>

　　var vm = new Vue({

　　　　el:'#app',

　　　　template:'<h1>hello</h1>',

　　　　beforeCreate:function(){

　　　　　　console.log('before create')

　　　　},

　　　　created:function(){

　　　　　　console.log('created')

　　　　},

　　　　beforeMount:function(){

　　　　　　console.log(this.$el);

　　　　　　console.log('before mount')

　　　　},

　　　　mounted:function(){

　　　　　　console.log(this.$el);

　　　　　　console.log('mounted')

　　　　}

　　})</script>

看到在beforeMount输出当dom是<div id='app'>hello world</div>，，在mounted输出的dom是<h1>hello</h1>这也印证了上面这张图的内容，在beforeMount的时候页面还没有被渲染，在mounted的时候页面已经被渲染完毕了

**beforeDestroy,destroyed**:

var vm = new Vue({

　　el:'#app',

　　template:'<h1>hello</h1>',

　　beforeCreate:function(){

　　　　console.log('before create')

　　},

　　created:function(){

　　　　console.log('created')

　　},

　　beforeMount:function(){

　　　　console.log(this.$el);

　　　　console.log('before mount')

　　},

　　mounted:function(){

　　　　console.log(this.$el);

　　　　console.log('mounted')

　　},

　　beforeDestroy:function(){

　　　　console.log('beforeDestroy')

　　},

　　destroyed:function(){

　　　　console.log('destroyed')

　　}

})

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

这张图上还有两个生命周期函数，叫做beforeUpdate和updated，这两个生命周期函数在什么时候执行呢

**beforeUpdate,updated**:

var vm = new Vue({

　　el:'#app',

　　template:'<h1>hello</h1>',

　　beforeCreate:function(){

　　　　console.log('before create')

　　},

　　created:function(){

　　　　console.log('created')

　　},

　　beforeMount:function(){

　　　　console.log(this.$el);

　　　　console.log('before mount')

　　},

　　mounted:function(){

　　　　console.log(this.$el);

　　　　console.log('mounted')

　 },

　　beforeDestroy:function(){

　　　　console.log('beforeDestroy')

　　},

　　destroyed:function(){

　　　　console.log('destroyed')

　　},

　　beforeUpdate:function(){

　　　　console.log('before updated')

　　},

　　updated:function(){

　　　　console.log('updated')

　　}

})

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

updated:当数据重新渲染之后，updated这个生命周期函数会被执行

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