1. 特点
   1. 采用增量式开发
   2. 用于构建用户界面的渐进式框架
   3. 响应的数据绑定(数据改变，页面自动更新)
   4. 可组合的视图组件
2. 介绍
   1. 指令带有前缀 v-，以表示它们是 Vue 提供的特殊属性
      1. eg: v-bind:title="message"(将这个元素节点的 title 属性和 Vue 实例的 message 属性保持一致”) (v-bind:可简写为:)(title= 后面可以写模版字符串,默认变量有i)
   2. v-if="seen"（元素显示）不渲染，不存在
   3. v-show 切换显示隐藏，（改变display），如果需要经常切换显示和隐藏，那么使用v-show,数据没有就不渲染结构使用v-if
   4. v-for="todo in todos"（v-for 指令可以绑定数组的数据来渲染一个项目列表）
   5. v-on:click="reverseMessage"（v-on 指令绑定一个事件监听器） （v-on可以简写为@）  
      @click = "事件处理函数（传参， $event）"模版中有一个全局的变量 $event  
      @click = 事件对象是事件处理函数的第一个参数
   6. <input v-model="message">(v-model 指令，它能轻松实现表单输入和应用状态之间的双向绑定。) （双向数据绑定（数据影响模版， 模版中的方法改变数据）
   7. MVVM
      1. M model 数据
      2. V view 视图
      3. VM view-model, 视图-模型
      4. view (DOM listener) model
      5. model (directives) view
   8. 事件修饰符 @keydown.13.37 （keydown.enter.left）回车和←键
3. 组件
   1. 组件命名规则
      1. 不能使用html自带的标签
      2. 声明组件时，使用烤串命名法和驼峰命名法都可以，但是使用的时候只能使用烤串命名法
   2. 组件中的data是一个函数，不是对象。
   3. 将组件标签之间的内容插入组件（插槽slot）
   4. 除非组件模版中包含一个slot标签，否则组件之间的内容将被舍弃
4. 使用vue-cli新建vue项目
   1. 工具：npm i vue-cli -g
   2. 命令行中 vue -V出现版本号安装成功
   3. vue init webpack project(项目名称)
   4. cd进入project
   5. 先安装模块 npm i
   6. 启动项目 npm run dev
5. vue-cli项目目录
   1. src
      1. main.js是整个应用的入口js，启动应用在这里完成
      2. App.vue 是整个个应用的根部件
   2. 以 .vue结尾的为单文件组件，
      1. template 存放模版，只能有一个根元素
      2. script 逻辑
         1. export default{  
            }
      3. style 样式
         1. sass
         2. less
      4. 一个组件使用另一个组件
         1. 引入组件 import{组件名（变量名）} from '组件地址'
            1. 组件名不能和html标签名相同
         2. 当使用另一个组件前必须在当前组件实例中注册  
            export default{

components: {

headerNav: header

headerNav

}

}

1. vuex的使用
   1. 安装vuex
      1. npm i vuex --save
   2. 使用：

import Vuex from 'vuex'

Vue.use(Vuex)

let store = new Vuex.Store({

state: {

// 定义状态

n：0

},

mutations :{

自定义事件名(state,要变的数){

state.n = 要变的数

}

}

})

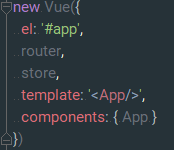
new Vue({

...

store

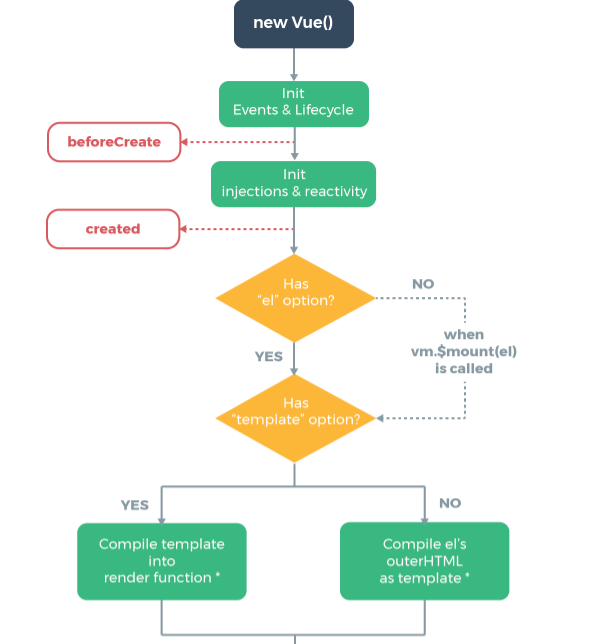
})

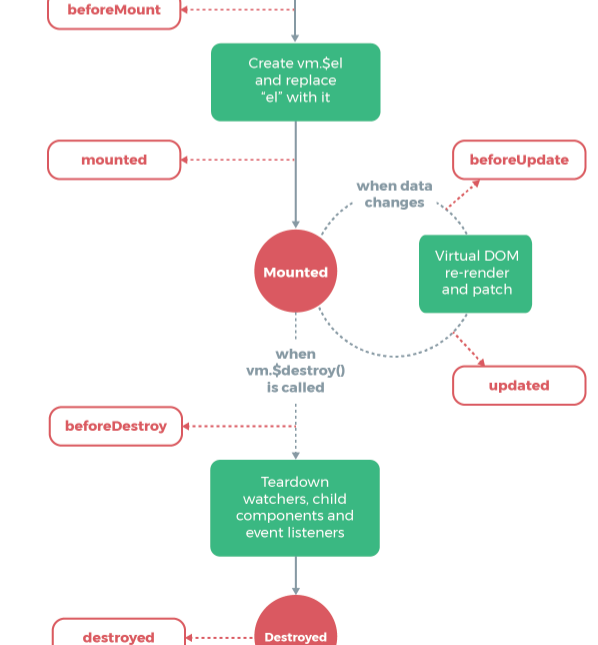
* 1. main.js 引用



* 1. 组件中取值 this.$store.state取值
  2. 改变state中的值
     1. this.$store.commit('自定义事件名', 数字)

1. 直接修改vuex中的state，vue开发工具记录不到
2. mutation是一个同步函数，使用mutation执行异步操作Vue的调试工具记录不到异步操作的结果
3. 所有异步操作放在action中，在action中提交Mutation
4. commit 一个Mutation， this.$store.commit('Mutation函数'，参数)
5. dispatch 一个Action， this.$store.dispatch('Action函数'，参数)
6. axios用法
   1. Axios.get('http://192.168.43.227:3000/info/').then(functon(pramas){  
       console.log(pramas)  
      })
7. express 设置允许跨域 header
8. vue-router可以管理路径和视图的映射关系
9. vue-router
   1. vue router的模式
      1. 默认是hash模式
         1. 地址栏中带上hash #/home #/about
      2. 切换为history模式
         1. 地址栏中 /home /about
10. veu router的标签
    1. <router-view/>在配置的配置项中，让路径对应的组件渲染在指定这个标签的位置
    2. <router-link/>生成链接，自动监听点击的事件，帮助渲染对应路径的组件
       1. 传入的props，
          1. to='路径'
          2. tag='标签名'生成的标签
          3. exact 精确匹配 只匹配当前激活的导航，添加样式
    3. \* 用与404NotFound
    4. redirect重定向
    5. 控制当前的激活状态的导航的样式，默认值： router-link-active，自定义名字 linkActiveClass
    6. 精确匹配当前激活状态的导航，默认值：router-exact-link-active 改名 linkExactActiveClass,
    7. 子路由 children
       1. path不带/ 视图嵌套路径嵌套
       2. path 带/ 视图嵌套路径不嵌套
    8. 动态路由
       1. path: '/user/:id?' :就是动态路径，id就类似变量一样，存访问路径的值 /user/,?代表可有可无。
       2. 组件内部可以通过this.$route拿到路由信息对象
       3. 在组件内部修改path，组件不会重新渲染，目的是达到复用的效果，但path改变了，用wath监控this.$route的变化，重新渲染。
11. 生命周期函数





* 1. 一般在created中使用ajax获取数据
  2. 一般在mounted中获取页面元素
  3. 挂载顺序，先子级后父级

1. [编程式导航](https://router.vuejs.org/zh-cn/essentials/navigation.html)
2. [导航守卫](https://router.vuejs.org/zh-cn/advanced/navigation-guards.html)（路由发生变化时触发）

位置：

1. 写在vueRouter的实例上

// 任何一个访问的路径，都会执行这个钩子函数

vueRouter.beforeEach((to,from,next) => {

to: 即将要进入的目标 路由对象

from: 当前导航正要离开的路由

})

1. 写在配置路由信息上

{

...

beforeEnter (to,from,next){

next()

}

}

1. 写在组件中

* beforeRouteEnter (to, from, next) {

// 在渲染该组件的对应路由被 confirm 前调用

// 不！能！获取组件实例 `this`

// 因为当守卫执行前，组件实例还没被创建

},

* beforeRouteUpdate (to, from, next) {

// 在当前路由改变，但是该组件被复用时调用

// 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

// 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

// 可以访问组件实例 `this`

},

* beforeRouteLeave (to, from, next) {

// 导航离开该组件的对应路由时调用

// 可以访问组件实例 `this`

}

完整的导航解析流程

1. 导航被触发。
2. 在失活的组件里调用离开守卫。
3. 调用全局的 beforeEach 守卫。
4. 在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。
5. 在路由配置里调用 beforeEnter。
6. 解析异步路由组件。
7. 在被激活的组件里调用 beforeRouteEnter。
8. 调用全局的 beforeResolve 守卫 (2.5+)。
9. 导航被确认。
10. 调用全局的 afterEach 钩子。
11. 触发 DOM 更新。
12. 用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。
13. redirect 第一个参数为 路由信息对象

过渡动画

1. 用transition包起来
2. 默认的class类是v-开头
3. name="fade"将v-开头改为fade-开头
4. .v-enter .v-enter-to .v-enter-active
5. .v-leave .v-leave-to .v-leave-active
6. 热加载文件
7. Vue中不能检测对象属性的添加和删除
8. 给组件添加Vue.set(vm.obj, 'age', 1000)
9. 给实例添加vm.$set(vm.obj, 'age', 1000)
10. 子级通知父级
    1. 子级this.$emit('remove')
    2. 父级@remove="removeHandle"