**知识点**

**导航守卫**

**router.beforeEach（to, from, next）**：全局前置守卫。

其中to：Route：即将要进入的目标路由对象；

form：Route：当前导航正要离开的路由，

next：Function：一定要调用该方法来resolve这个钩子。

本项目用来做用户权限限制

**App.vue**

**router-view**：是一个顶级的外链。它会渲染一个和顶级路由匹配的组件。

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。使用如下：

父组件：provide() {

return {

map\_nodeObj: { map\_node: this.obj }

}

}

子组件：inject:{

map\_nodeObj:{

default: () => {

return { map\_node: '0' }

}

}

运行顺序

data

provide

created // 在这个阶段$el还未生成，在这先处理privide的逻辑，子孙组件才可以取到inject的值

mounted

......

data：app对象的一个属性，有三种写法

1.data() {

return {count: 0}

},【ES6写法】

2.data: {

count: 0

}

3.data: function() {

return {

count

}

}

因为app对象不会被复用，当在app多出同时调用同一组件，就会导致该组件共享data，所以使用第三种方式返回一个函数。

methods：app对象中的方法属性。

**组件**

scoped：私有化组件样式。穿透方式：

1.<style scoped>

外层 >>> 第三方组件 {

样式

}

</style>

2.在定义一个style不含scoped

**使用组件：**

导入 import appHeader from '@/components/appHeader/appHeader.vue';

注册 components: {

'app-header' : appHeader,

}

使用 <app-header></app-header>

computed：属性会基于它所依赖的数据进行缓存，如果所依赖的data值没有改变就不会再次计算。

watch：当你需要在数据变化响应时，执行异步操作，或高性能消耗的操作，自定义 watcher 的方式就会很有帮助。

**生命周期：**

**beforeCreate:** function(){

console.log('beforeCreate:刚刚new Vue()之后，这个时候，数据还没有挂载呢，只是一个空壳')

console.log(this.msg)//undefined

console.log(document.getElementsByClassName("myp")[0])//undefined

},

**created**: function(){

console.log('created:这个时候已经可以使用到数据，也可以更改数据,在这里更改数据不会触发updated函数')

this.msg+='!!!'

console.log('在这里可以在渲染前倒数第二次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取')

console.log('接下来开始找实例或者组件对应的模板，编译模板为虚拟dom放入到render函数中准备渲染')

},

**beforeMount**: function(){

console.log('beforeMount：虚拟dom已经创建完成，马上就要渲染,在这里也可以更改数据，不会触发updated')

this.msg+='@@@@'

console.log('在这里可以在渲染前最后一次更改数据的机会，不会触发其他的钩子函数，一般可以在这里做初始数据的获取')

console.log(document.getElementsByClassName("myp")[0])//undefined

console.log('接下来开始render，渲染出真实dom')

},

// render:function(createElement){

// console.log('render')

// return createElement('div','hahaha')

// },

**mounted**: function(){

console.log('mounted：此时，组件已经出现在页面中，数据、真实dom都已经处理好了,事件都已经挂载好了')

console.log(document.getElementsByClassName("myp")[0])

console.log('可以在这里操作真实dom等事情...')

// this.$options.timer = setInterval(function () {

// console.log('setInterval')

// this.msg+='!'

// }.bind(this),500)

},

**beforeUpdate**: function(){

//这里不能更改数据，否则会陷入死循环

console.log('beforeUpdate:重新渲染之前触发')

console.log('然后vue的虚拟dom机制会重新构建虚拟dom与上一次的虚拟dom树利用diff算法进行对比之后重新渲染')

},

**updated**: function(){

//这里不能更改数据，否则会陷入死循环

console.log('updated:数据已经更改完成，dom也重新render完成')

},

**beforeDestroy**: function(){

console.log('beforeDestory:销毁前执行（$destroy方法被调用的时候就会执行）,一般在这里善后:清除计时器、清除非指令绑定的事件等等...')

// clearInterval(this.$options.timer)

},

**destroyed**: function(){

console.log('destroyed:组件的数据绑定、监听...都去掉了,只剩下dom空壳，这里也可以善后')

}

})