# 五 导航过度动画

## 1 基本用法

vue提供了transition封装组件，来添加过渡动画，通过添加删除css类名或者钩子函数来控制。

常见CSS类名：

v-enter 进入过渡的开始状态

v-enter-active 进入活动状态

v-enter-to 既然怒的结束状态

v-leave

v-leave-active

v-leave-to

基本用法：

|  |
| --- |
| template部分：  <**transition**>  <**router-view**></**router-view**> </**transition**>  style部分：  .**v-enter**{  **opacity**: 0; } .**v-enter-to**{  **opacity**: 1; } .**v-enter-active**{  **transition**: 1**s**; } |

当然还有很多其他的动画，非v-开头，使用方式如下：

|  |
| --- |
| <**transition name="left"**>  <**router-view**></**router-view**> </**transition**>  .**left-enter**{  **transform**: **translateX**(100%); } .**left-enter-to**{ //默认值可以不写  **transform**: **translateX**(0); } .**left-enter-active**{  **transition**: 1**s**; } |

## 2 过渡模式

当我们在切换导航时，可能切换动画还未完成，又点击了别的导航，这样会造成不同切换之间的动画互相重叠的问题。这时候需要使用过渡模式。

过渡模式：

in-out：新元素先进行过渡，完成之后当前元素过渡离开

out-in：当前元素先进行过渡，完成之后新元素过渡进入

|  |
| --- |
| <**transition mod="in-out"**>  <**router-view**></**router-view**> </**transition**> |

# 六 路由元信息

在路由配置中meta可以配置一些数据，用在路由信息对象中，访问meta中的数据方式：$route.meta

|  |
| --- |
| {  **path**: **'/index'**,  **component**: myindex,  **meta**: {  **subIndex**: 1  } }, |

使用：

|  |
| --- |
| **watch**: {  $route (to, **from**) {  ***console***.log(to.meta.**subIndex**)  ***console***.log(**from**.meta.**subIndex**)  } }, |

其中，to和from分别进入、离开的路由对象。

# 七 钩子函数

## 1 基本使用

|  |
| --- |
| Vue.use(Router)  **let** router = **new** Router({  ......... }) router.beforeEach((to, from, next) => {  **if** (to.**meta**.login === **true**) {  next(**'./login'**)  } **else** {  next()  } }) **export default** router |

如果不写next() 或者next的参数写为false，将会不渲染组件内容。

常用的使用案例：当一个页面登陆后才会让用户访问，这时候用户访问路由，我们可以使用next()进行重定向到登录页。

其他案例，比如我们在进入一个导航后，网页的title需要改变为当前页面需要的标题，那么我们可以使用afterEach这个导航钩子函数。（注意：title不能直接获得，需要完整的写：window.document.title）

## 2 组件级钩子函数

以上的钩子函数写在了路由外部，作用在了全局，如果在单个路由中的书写钩子函数，这个钩子函数只作用于该留有，当然也有组件级的钩子函数，书写在具体的组件script中。

|  |
| --- |
| <**script**>  **export default** {  data () {  **return** {  **test**: **'改变前'** }  },  beforeCreate () {  ***console***.log(**'beforeCreate'**)  },  beforeRouteEnter (to, **from**, next) {  next((vm) => {  vm.**test** = **'改变了'** })  }  } </**script**> |

执行顺序是：全局》路由》组件

## 3 常见钩子函数

router实例：

beforeEach afterEach

单个路由：

beforeEnter

组件级钩子：

beforeRouteEnter beforeRouteUpdate beforeRouteLeave

钩子函数的参数：

to：要进入的目标路由对象，到哪里去

frome：正要离开的导航路由对象，从哪里来

next：决定是否跳转或者取消导航

# 十 Axios



### 1 Axios简介





### 2 简单使用

安装模块 axios与vueaxios后，在App.vue中使用axios：

|  |
| --- |
| <**script**> **import** axios **from 'axios' export default** {  **name**: **'app'**,  created(){  axios({  **method**: **'get'**,  **url**: **'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'** })  .then((res)=>{  ***console***.log(res.**data**)  })  .catch((err)=>{  ***console***.log(err)  })  } } </**script**> |

注意：axios也提供了直接使用axios.get(url)这样的方式发送不同类型的请求。

更简洁的方式：在main.js中全局使用：

|  |
| --- |
| **import** Axios **from 'axios' import** VueAxios **from 'vue-axios'** Vue.use(VueAxios,Axios) |

在任意的组件中都可以使用 this.$http.get(url)来发送请求。

### 3 参数传递

get请求参数

|  |
| --- |
| axios.get(**'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'**,{  **param**: {  **abc**: **'123'** } }) |

post请求参数

|  |
| --- |
| axios.post(**'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'**,{  **abc**: **'123'** }) |

### 4 自定义请求





其他配置：  


需要return，没有return会是undefined

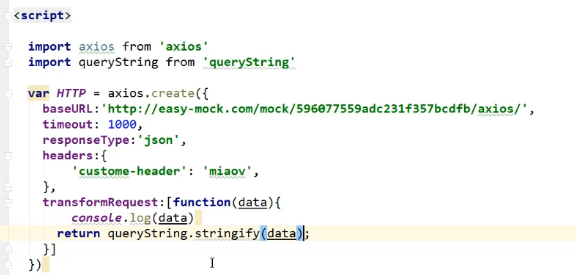


transform可以用来转换发送的数据格式

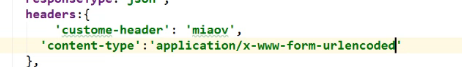
比如我们在请求时，请求数据是json形式：



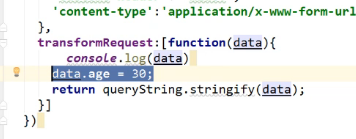
我们想使用字符串 miaov=ketang&username=leo这种格式，可以在transform中修改：



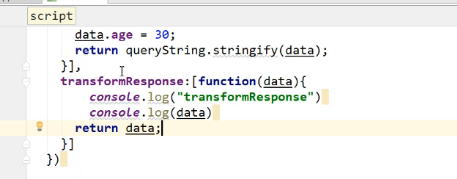
我们还需要设置头信息，让post请求支持这种字符串格式：



发送之前还可以对数据进行额外处理，如改变值：



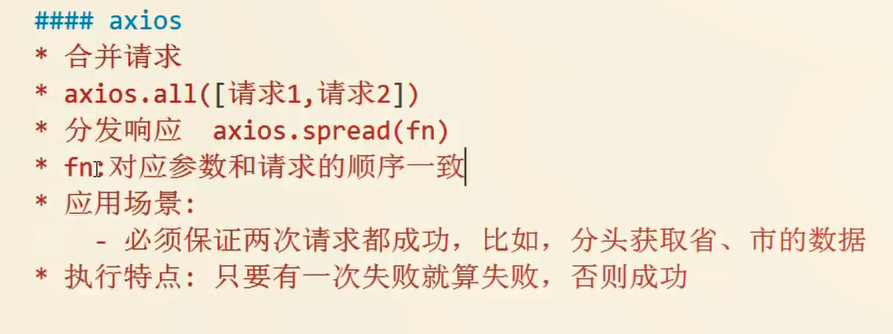
我们还可以请求结果进行加工：



### 5 取消请求配置



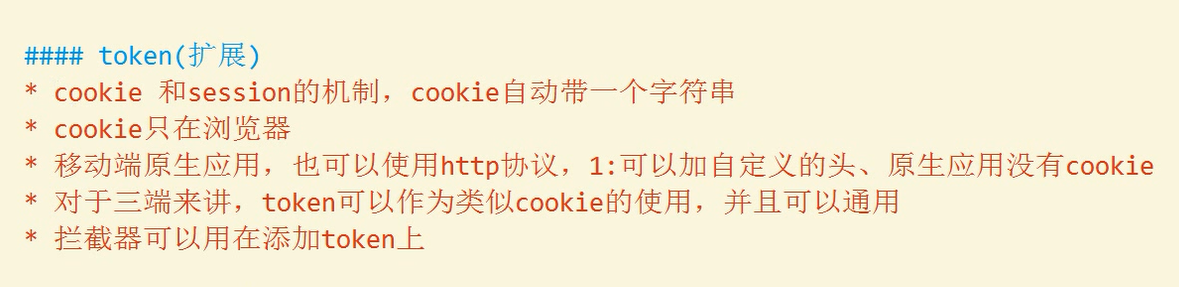
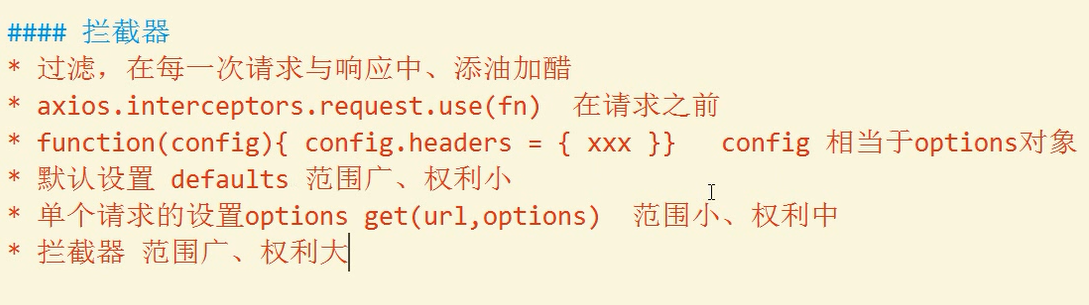
### 6合并请求

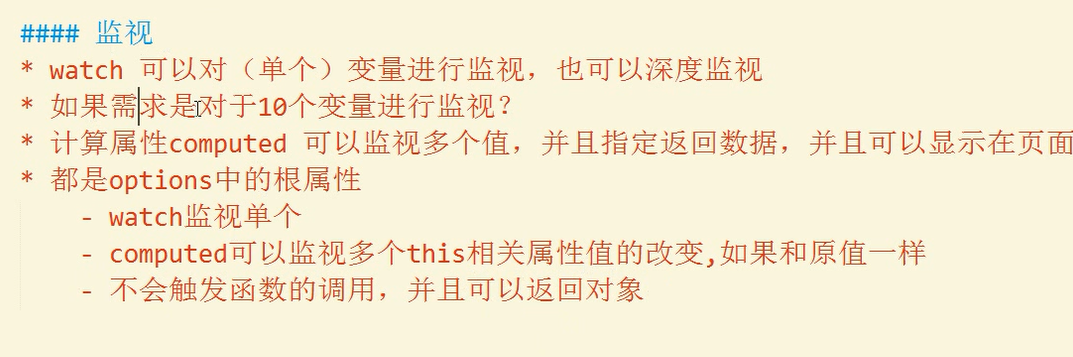


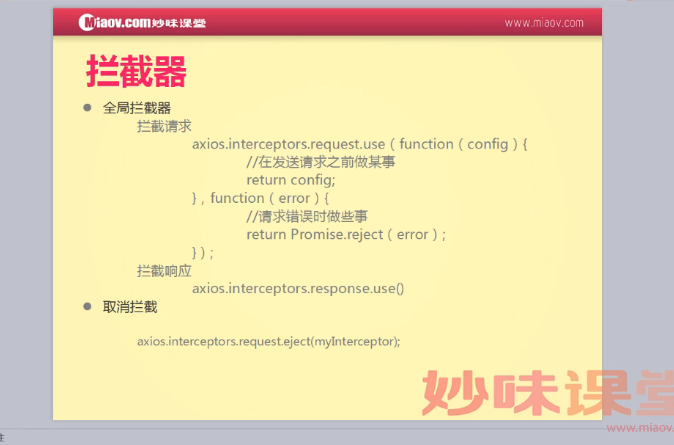
### 7 拦截器

我们可以对请求和相应进行拦截

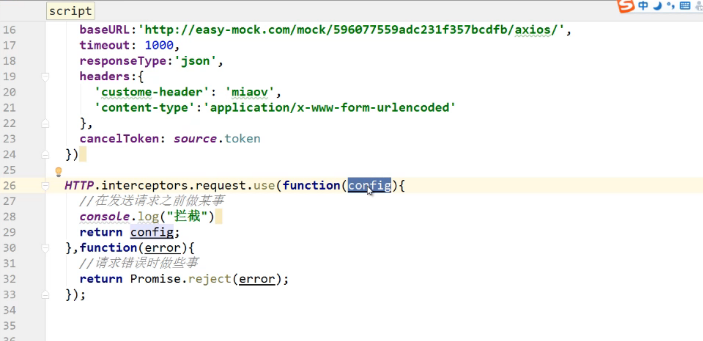
注意：拦截器是在发起请求后，真正开始向服务器发送时才执行，所以之前组件内，方法内自己设置的独立的请求配置都会失效，所以拦截器的权限是最大的。





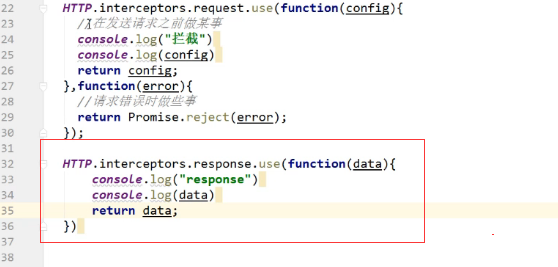


请求拦截案例：



只有return config 后才会发哦送你个请求

响应拦截案例：

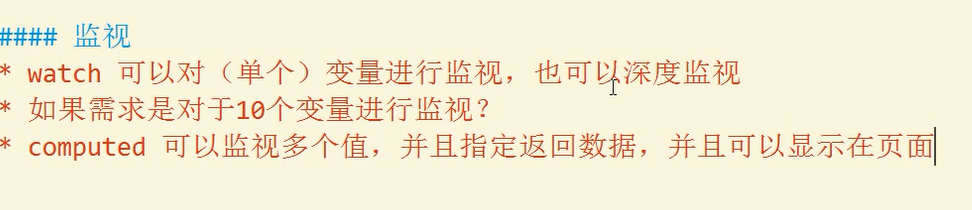


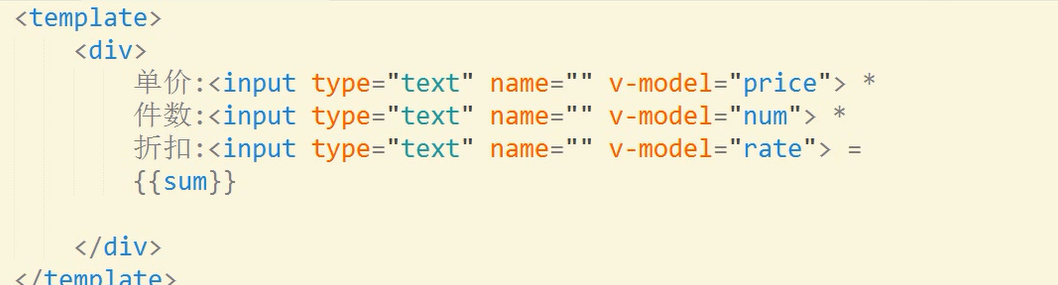
# 知识点

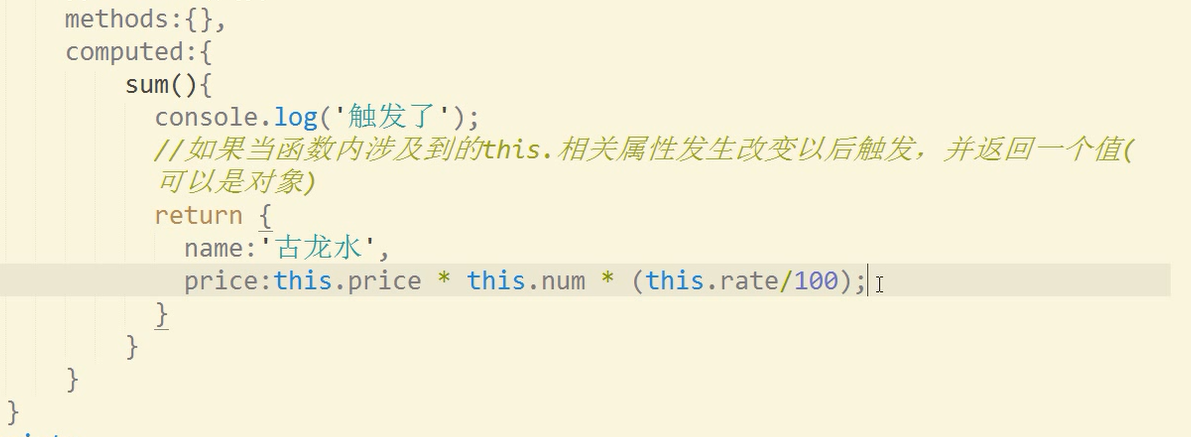
0 深度监视



普通数据直接wathc即可，对象内仍然有属性，需要按照上述设置进行深度监视。







## 1 $nextTick 异步更新队列

<div id="app">  
 <div id="div" v-if="showDiv">文本</div>  
 <button @click="getText">点击获取文本内容</button>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 new Vue({  
 el: '#app',  
 data: {  
 showDiv: false  
 },  
 methods:{  
 getText: function () {  
 this.showDiv = true;  
 let text = document.getElementById('div').innerHTML;  
 console.log(text);  
 }  
 }  
 });  
</script>

示例代码无法获取到文本内容，且报错：cannot read property ‘innerHTML’。

vue观察到数据时，并不是直接更新DOM，而是开启一个队列，缓冲在同一个事件循环中发生的所有数据变化，缓冲时会去除重复数据。在下一个事件循环tick中，刷新队列并执行更新。所以如果用一个for循环来动态改变数据100次，其实只会应用最后一次。

vue会依据当前浏览器环境优先使用源生的Promise.then和MutationObserver，如果都不支持，使用setTimeout代替。

上述案例中，执行this.showDiv=true时候，div还没被创建，直到下一个vue事件循环，才开始创建。$nextTick可以知道什么时候DOM更新完成。

getText: function () {  
 this.showDiv = true;  
 this.$nextTick(function () {  
 let text = document.getElementById('div').innerHTML;  
 console.log(text);  
 });  
}

## 2 X-Templates定义模板

如果不使用webpack、gulp，那么template模板的书写是一场噩梦。vue支持在script表使用text/x-template类型，并指定id，将id赋值给template。此时可以愉快的书写html，不用考虑换行问题。

<script type=”text/x-template” id=”my-comp”>

//书写html模板

</script>

//在vue中的使用

template: ‘#my-comp’

## 3 手动挂载实例

一般我们通过new Vue()创建vue实例，如果需要动态创建，需要使用Vue.extend和$mount两个方法手动挂载实例。

vue在实例化时，没有el，那么会处于未挂载状态，没有关联的dom元素，使用$mount手动挂载，这个方法返回实例本身，因而可以链式调用。

<div id="app">  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let MyComponent = Vue.extend({  
 template: '<div>hi</div>'  
 });  
 new MyComponent().$mount('#app');  
</script>

也可以写为：

new MyComponent({

el:’#app’

});

## 4 自定义指令

可以在选项对象的directives属性中添加：

|  |
| --- |
| **directives**: {  **'focus'**: {  update(el,binding){  **if**(binding.**value**){  el.focus();  }  }  } },  在标签中直接使用v-focus即可 |

## 5 watch

在上述案例中，我们从loacalStorage中存储数据，在界面中，依据业务需要，对该数据进行增删改，分别对应了vue的method中的增删改方法，但是如果一旦数据需要变动，增删改三个方法都要进行修改，这是很麻烦的。这时候我们可以使用watch来统一监控属性改变。



var list = store.fetch(‘**new-calss**’);

|  |
| --- |
| **watch**: {  list: **function** () {  store.save(**'new-calss'**,**this**.list);  }  }, |

此时只要list发生变化，就会执行上述function。无需再在 编辑，插入，删除等方法中进行数据存储。

但是这样无法对list进行深度监控，即当list对象内部属性改变的时候，仍然无法监控到，下列方法可以实现深度监控：

|  |
| --- |
| **watch**: {  **list**: {  handler: **function** () {  *//存储数据* },  **deep**: **true** } }, |

## 8 模板render函数

### 8.1 render函数初体验

以下代码：

<div id="app">  
 <mycomp :level="2" title="特性">特性</mycomp>  
 <script type="text/x-template" id="my-comp">  
 <div>  
 <h1 v-if="level === 2"><a :href="'#' + title"><slot></slot></a></h1>  
 <h1 v-if="level === 3"><a :href="'#' + title"><slot></slot></a></h1>  
 <h1 v-if="level === 4"><a :href="'#' + title"><slot></slot></a></h1>  
 </div>  
 </script>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('mycomp',{  
 template: '#mycomp',  
 props: {  
 level: {  
 type: Number,  
 required: true  
 },  
 title: {  
 type: String,  
 default: ''  
 }  
 }  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

使用render的写法：

<div id="app">  
 <mycomp :level="2" title="特性">特性</mycomp>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 Vue.component('mycomp',{  
 template: '#mycomp',  
 props: {  
 level: {  
 type: Number,  
 required: true  
 },  
 title: {  
 type: String,  
 default: ''  
 }  
 },  
 render: function (createElement) {  
 return createElement(  
 'h' + this.level,  
 [  
 createElement(  
 'a',  
 {  
 domProps: {  
 href: '#' + this.title  
 }  
 },  
 this.$slots.default  
 )  
 ]  
 );  
 }  
 });  
 new Vue({  
 el: '#app'  
 });  
</script>

render函数通过createElement参数创建虚拟dom。（访问slot的使用场景在render函数里）

常见属性有：

class:{} 绑定class，和v-bind:class 一样

style:{} 绑定样式，和v-bind:style一样

attrs:{} 添加行间属性

domProps:{} DOM元素属性

on:{} 绑定事件

nativeOn:{} 监听原生事件

directives:{} 自定义指令

注意：domPropps属性中可以直接书写dom对象本身的方法，如：

domPropps:{

innerHTML: ‘<h1>test</h1>’

}

这里要谨慎使用，因为他会替换掉createElement中的元素。