1. **导航/路由守卫**

**导航表示路由正在发生改变**

**1.全局前置守卫**

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

...

next();

});

to: Route: 即将要进入的目标路由对象

from: Route: 当前导航正要离开的路由

全局前置守卫中必须设置next()

目前在项目中有一些路由信息只能在用户登录后才可访问，而还有一些路由信息无论是用户登录与否都可以访问，所以目前就不能直接使用VUEX/WebStorage进行统一的条件判断了！所以只能在需要用户登录才能访问的路由上添加meta元信息，这样就可以区分哪些路由需要用户登录了！代码如下：

{

    path:'/register',

    component:Register

  },

  {

    path:'/me',

    component:Me,

    //在路由的头部添加元信息

    meta:{

      //requiresAuth是自定义的头部参数名称,

      //标识访问该路由时是否进行用户登录的检测

      requiresAuth:true

    }

  }



接下来就可以通过全局前置守卫进行检测，代码如下：



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

  //console.log(to);

  //console.log(to.meta.requiresAuth);

  //检测目标路由信息是否包含requiresAuth的元信息

  //如果包含则代表需要进行用户登录状态的检测

  if(to.meta.requiresAuth){

    //检测用户是否登录了?

    if(store.state.author.isLogined == false || sessionStorage.getItem('isLogined') == false){

      //强行跳转到登录路由(附加目标路由信息作为URL地址栏的参数)

      //该参数将在用户登录成功后，再次跳转回该路由

      router.push('/login?redirect=' + to.fullPath);

    } else {

      //访问用户期望访问的信息(如我的,也就是说访问那些用户已登录后才能访问的路由)

      next();

    }

  } else {

    //不需要用户登录即可访问的路由

    next();

  }

});

**router代表路由实例（也称路由表或路由器）**

**route代表路由对象（URL地址）**

1. **Vuex中的辅助函数**

* **mapState()**

mapState()函数用于为组件创建计算属性，以返回Vuex Store中的状态，其语法结构是：

mapState(array|object)

示例代码如下：

<script>

import {mapState} from 'vuex';

export default {

  computed: {

    ...mapState([

      'username',

      'age',

      'sex',

      'salary',

      'products'

    ])

  },

}

</script>



* **mapGetters()**

mapGetters()函数用于为组件创建计算属性以返回vuex getters中的返回值，其语法结构是：

mapGetters(array|object)

示例代码如下：

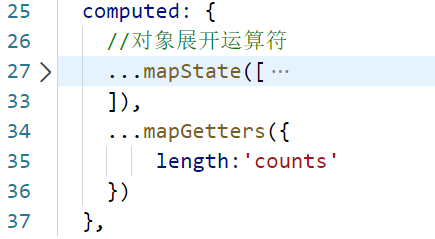
computed: {

   ...mapGetters({

        length:'counts'

    })

  }



* **mapMutations()**

mapMutations()函数用于创建组件方法以提交mutation，其语法结构是：

mapMutations(array|object)

示例代码如下：

<script>

import {mapMutations} from 'vuex';

export default {

  methods: {

    ...mapMutations([

      'sAdd'

    ]),

    handler(){

      //构建"商品"对象

      var obj = {

        productName:this.productName,

        salePrice:this.salePrice

      };

      //调用Vuex的Mutations

      //this.$store.commit('sAdd',obj);

      this.sAdd(obj);

      this.$router.push('/access1');

    }

  }

}

</script>



* **mapActions()**

mapActions()函数用于创建组件方法以分发Action，其语法结构是：

mapActions(array|object)

示例代码如下：

<script>

import {mapState,mapActions} from 'vuex';

export default {

  computed: {

    ...mapState([

      'data'

    ])

  },

  methods: {

    ...mapActions([

      'getData'

    ]),

    handler(){

        //this.$store.dispatch('getData');

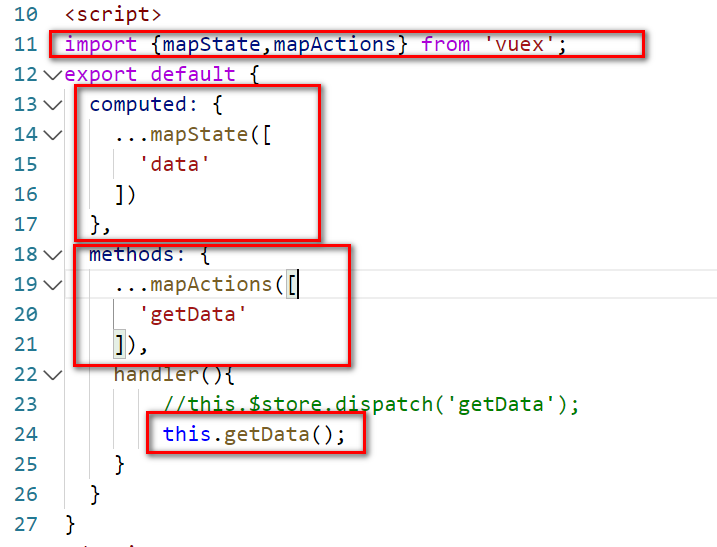
        this.getData();

    }

  }

}

</script>



* 请简述VUEX的工作流程

1. **Actions发送异步请求到服务器，并且接收服务器返回的数据**
2. **Actions接收到数据后commit到Mutations,然后Mutations操纵state的数据**
3. **state数据发生变化后通知相关的组件,组件数据发生变化。**



1. **HTML5新特性 -- 拖放**

**拖放是HTML5的新特性,即抓取一个对象从一个位置到另一个位置。**

**在HTML5中任何元素都可以拖放。**

**为了解决浏览器的兼容性问题，建议在拖放在文本类型的元素添加draggable="true"**

1. **DragEvent接口**

**DragEvent接口是表示拖、放事件的接口，该接口继承自MouseEvent接口，在DragEvent接口中包含的事件有：(源对象事件)**

* + - **dragstart，该事件在源对象开始被拖动时触发，其语法结构是：**

HTMLElement.ondragstart = function(){

...

}

* + - **drag，该事件在源对象拖动过程中触发，其语法结构是：**

HTMLElement.ondrag = function(){

...

}

* + - **dragend，该事件在源对象被拖动结束后触发（也可能在目标区域内，也可能在目标区域外），其语法结构是:**

HTMLElement.ondragend = function(){

...

}

**目标对象事件**

* + - **dragenter，该事件在源对象进入目标对象时触发，其语法结构是：**

HTMLElement.ondragenter = function(){

...

}

* + - **dragover，该事件在源对象在目标对象上悬停时触发，其语法结构是：**

HTMLElement.ondragover = function(event){

event.preventDefault();

...

}

**注意：dragover事件中必须执行event.preventDefault()，否则不会触发drop事件!!!**

* + - **dragleave，该事件在源对象拖离目标对象时触发，其语法结构是：**

HTMLElement.ondragleave = function(){

...

}

* + - **drop，该事件在源对象在目标对象上释放时触发，其语法结构是：**

HTMLElement.ondrop = function(event){

event.preventDefault();

...

}

**事件的触发顺序:**

**dragstart -> drag -> dragenter -> dragover -> drop -> dragend**

1. **dataTransfer属性**

dragEvent接口的dataTransfer属性用于保存拖放数据，将返回DataTransfer对象，其语法结构是：

DataTransfer DragEvent.dataTransfer

1. **DataTransfer对象**
   * + setData()方法

setData()方法用于为一个给定的类型设置数据，如果类型不存在，则自动添加到未尾，如果存在的话，则替换原来的数据，其语法结构是：

DataTransfer.setData(类型,数据)

* + - getData()方法

getData()方法用于返回指定类型的数据，如果类型不存在或者没有数据，则返回空字符串，其语法结构是：

variable = DataTransfer.getData(类型)

**逻辑删除的实现:**

**需要在对应的数据表中添加一个标识字段,用于标识**

**记录的状态,而且该字段只有两个状态,假设0表示正常状态,1表示逻辑删除状态，另外逻辑状态的改变是通过UPDATE语句实现。如站长推荐，是否热卖等都是如法炮制。**

1. **Node中实现文件上传**
2. **标准的表单文件上传**

**此时表单必须设置以下属性：**

* + **表单的提交方式必须为POST**
  + **必须设置表单的enctype="multipart/form-data"属性**

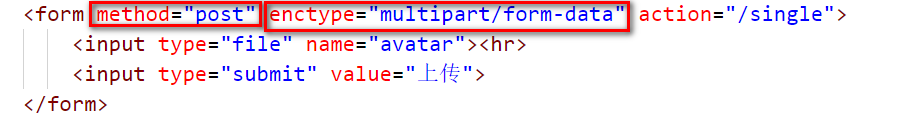
**示例代码如下：**

  <form method="post" enctype="multipart/form-data" action="/single">

      <input type="file" name="avatar"><hr>

      <input type="submit" value="上传">

  </form>



1. **Multer中间件**

Multer 是一个 node.js 中间件，用于处理 multipart/form-data 类型的表单数据，它主要用于上传文件。

安装

npm install --save multer

<https://github.com/expressjs/multer/blob/master/doc/README-zh-cn.md>