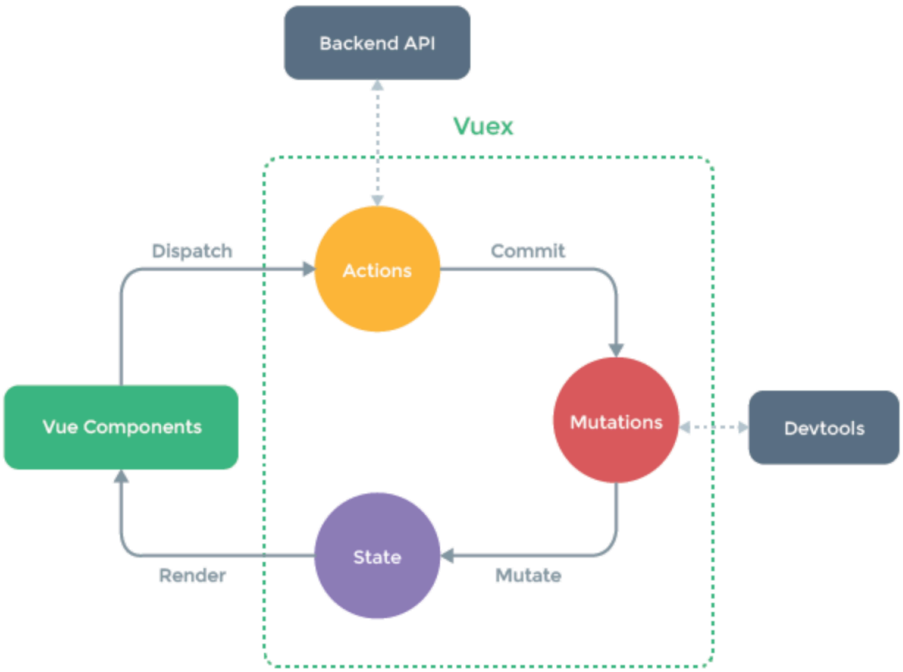
学习记录

1. 学习vuex



vuex中的主要为五部分 state mutations actions

state: 为主要存储共有变量信息，一般mutations中使用的变量最后都进行初始化

mutations: 只能进行同步操作 异步操作无法使用devtools捕获到信息 可以直接更改state数据

一般使用const MUTATION = "MUTATION" 这种方式定义名称

使用 store.commit ("MUTATION",payload) 来执行(payload只参数)

actions: 可以进行异步操作 而不能直接更改state数据 只能更改操作mutations

```
action:{
    motation:function(context){
        context.commit("MUTATION",payload)；//这里context相当于store的一个副本
    }
}
```

使用store.dispatch("action") 进行相应actions调用

getters: 主要是对state中的数值进行一些处理 类似于computed计算值属性 通过store.getters.getName（state,getters）调用

module: 模块化 为了更好的管理vuex中的数据，当数据量达到一定值时会出现混乱的情况 这个时候我们就可以使用module模块化来进行分级处理 让结构更加清晰

```js
const moduleA = {
  state: { ... },
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: { ... },
  mutations: { ... },
  actions: { ... }
}

const store = new Vuex.Store({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> moduleA 的状态
store.state.b // -> moduleB 的状态
```

模块化类似于我们声明函数的解构 像moduleA中的mutations参数state就是局部state 和我们函数是一个道理 而根节点为rootState
而为了更高的封装度 一般会使用命名空间namespace:true 属性 来为子module添加对应梯度路径名称
getters方法如果想要获取到根state 和getter 第三个四个参数可以使用rootState rootGetters

```js
modules: {
  foo: {
    namespaced: true,

    getters: {
      // 在这个模块的 getter 中，`getters` 被局部化了
      // 你可以使用 getter 的第四个参数来调用 `rootGetters`
      someGetter (state, getters, rootState, rootGetters) {
        getters.someOtherGetter // -> 'foo/someOtherGetter'
        rootGetters.someOtherGetter // -> 'someOtherGetter'
      },
      someOtherGetter: state => { ... }
    },

    actions: {
      // 在这个模块中， dispatch 和 commit 也被局部化了
      // 他们可以接受 `root` 属性以访问根 dispatch 或 commit
      someAction ({ dispatch, commit, getters, rootGetters }) {
        getters.someGetter // -> 'foo/someGetter'
        rootGetters.someGetter // -> 'someGetter'
```

局部=》全局触发： 在module模块局部actions中想要触发全局空间的actions 和mutations只需要在第三个参数传入{ root:true} 如下图

```
    getters: {
      // 在这个模块的 getter 中，`getters` 被局部化了
      // 你可以使用 getter 的第四个参数来调用 `rootGetters`
      someGetter (state, getters, rootState, rootGetters) {
        getters.someOtherGetter // -> 'foo/someOtherGetter'
        rootGetters.someOtherGetter // -> 'someOtherGetter'
      },
      someOtherGetter: state => { ... }
    },

    actions: {
      // 在这个模块中， dispatch 和 commit 也被局部化了
      // 他们可以接受 `root` 属性以访问根 dispatch 或 commit
      someAction ({ dispatch, commit, getters, rootGetters }) {
        getters.someGetter // -> 'foo/someGetter'
        rootGetters.someGetter // -> 'someGetter'

        dispatch('someOtherAction') // -> 'foo/someOtherAction'
        dispatch('someOtherAction', null, { root: true }) // -> 'someOtherAction'

        commit('someMutation') // -> 'foo/someMutation'
        commit('someMutation', null, { root: true }) // -> 'someMutation'
      },
      someOtherAction (ctx, payload) { ... }
    }
  }
}
```

局部=》全局声明： 在声明的action中添加root:true 属性 如下图

```js
{
  actions: {
    someOtherAction ({dispatch}) {
      dispatch('someAction')
    }
  },
  modules: {
    foo: {
      namespaced: true,

      actions: {
        someAction: {
          root: true,
          handler (namespacedContext, payload) { ... } // -> 'someAction'
        }
      }
    }
  }
}
```

2. 学习ES6中的promise对象
结构如下 一般使用的场景为异步操作
promise 有pending （进行中）fulfilled(成功) rejected(失败三种状态)
而Promise中需要传递回调函数 这个回调函数有两个参数 一个是resolve（成功） 一个是reject（失败）
const promise = new Promise((resolve,reject)=>{

})
promise对象有then属性来处理resolve reject之后结果的处理，而resolve reject可以继续传递promise对象（这就意味着需要等待传递的那个promise执行完毕才会执行 而这种场景在开发中经常用到）
promise.then(
    function(value){ },
    function(error){ } //可省略
))

promise中的resolve 和reject执行并不会终止promise

```
new Promise((resolve, reject) => {
  resolve(1);
  console.log(2);
}).then(r => {
  console.log(r);
});
// 2
// 1
```

所以一般情况我们需要在前面加上return 来进行终止

```
new Promise((resolve, reject) => {
  return resolve(1);
  // 后面的语句不会执行
  console.log(2);
})
```

then方法可以返回promise对象 所以意味着可以进行链式操作 而后面都需要前面执行完毕才会执行这也是我们开发中会遇到的情况

而promise.prototype.catch 等于 .then(null , function(){}) 是在rejected的情况下触发的函数

已解决

1. 解决tomcat disconneted 端口占用错误

   lsof -i:port kill PID 结束端口占用程序重启tomcat

2. 解决webpack打包供应链平台样式覆盖问题

情况：修改element样式更改为设计图的样式，打包前样式完全替换覆盖成功，打包后element自带默认样式将替换样式重新覆盖

原因：main.js 中引入app.vue 和 element组件和css顺序出现错误导致同优先级的样式无法生效

3.vuex学习中对于模块化中的action暴露问题context不是内部的声明的 如何暴露出来？如何将mutation、getter声明到全局？

同样，对于模块内部的 action，局部状态通过 `context.state` 暴露出来，根节点状态则为 `context.rootState`：

```js
const moduleA = {
  // ...
  actions: {
    incrementIfOddOnRootSum ({ state, commit, rootState }) {
      if ((state.count + rootState.count) % 2 === 1) {
        commit('increment')
      }
    }
  }
}
```

理解错了 暴露出来就是单纯的可以调用，理解成局部声明为全局了

未解决

1.