vuex 和 async/await 使用说明

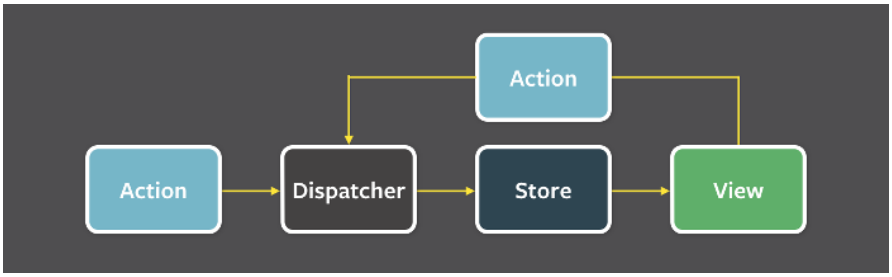
更新时间：2019-08-09 10:16:49

> 梦想只要能持久，就能成为现实。我们不就是生活在梦想中的吗？
>
> ——丁尼生

这一节将会介绍2个知识点，分别是 vuex 和 async/await，这两个知识点在后续的代码开发中会经常用到，所以掌握好是必备基础，下面将以代码和页面开发相结合的方式来讲解。

vuex

还记得上一节，我们在代码里使用了 this.$store.dispatch('setUser', resp.data) 来存储登录成功用户的数据，为了是我们后续在其他页面使用时，会比较方便的从store拿到，这里就用到了vuex。

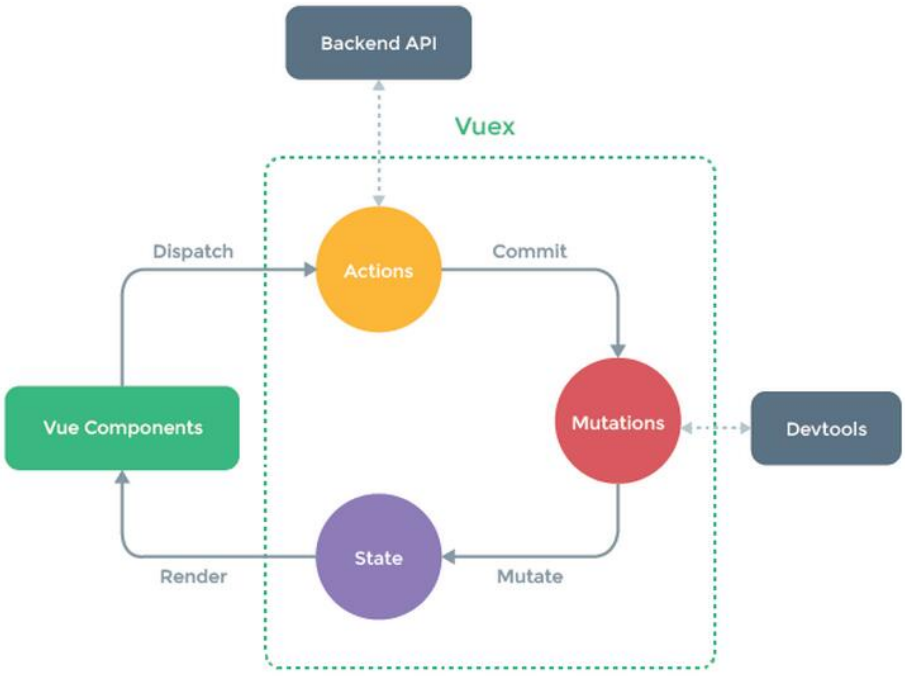


上图展示了 vuex 的主要工作流程，它的主要作用是将一些需要跨页面，跨组件的数据进行管理，这样我们在一个页面调用另外一个页面的数据时就可以从 vuex 的 store 里面获取，我们以设置登录用户的数据为例子，代码如下：

在前端项目的 store.js 里的新增逻辑：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    // 存储当前用户的数据
    currentUser: window.localStorage.getItem('cuser') ? JSON.parse(window.localStorage.getItem('cuser')) : {},
  },
  mutations: {
    /*
    * 设置当前用户mutations
    */
    currentUser (state, user) {
      state.currentUser = user
      //将当前用户数据储存在localStorage里
      window.localStorage.setItem('cuser', JSON.stringify(user))
    },

  },

  actions: {
    /*
    * 设置当前用户actions
    */
    setUser (context, user) {
      //增加action
      context.commit('currentUser', user)
    },

  }
})
```

代码里的`mutations`， `actions`，和`state`可以从下面的流程图来理解：



1. 当两个组件要通信时，首先需要有一个状态来表示，这个状态写在`state`里面。
2. 和`state`关联的是`mutations`，只有`mutations`可以去修改`state`的值，`mutations`和`state`一般是一一对应的。
3. 想要去调用`mutations`就需要有一个`action`来触发，所以需要在`actions`里去调用`commit`来触发`mutations`。

4. 最后就是调用 dispatch 去触发 action，注意调用 dispatch 是在vue的组件里，而 mutations， actions，和 state 的代码是写在 store 里面的，我们在不同的组件里调用 dispatch，通过一圈最终会到另一个组件里去触发更新。

我们在组件里调用 action，代码如下：

```
this.$store.dispatch('setUser', resp.data)
```

我们在组件里使用vuex里的数据：

```
computed: {
  //通过计算属性来获取store的数据
  myUser () {
    return this.$store.state.currentUser
  }
},
```

computed 表示计算属性，使用方法和 data 类似，但是对于一个属性来说，最好只在两个里面的其中1个里面去定义，避免重复。那么 computed 到底在什么时候去使用呢：

- computed 用来监控自己定义的变量，该变量不在 data 里面声明，直接在 computed 里面定义，然后就可以在页面上进行双向数据绑定展示出结果或者用作其他处理。
- computed 比较适合对多个变量或者对象进行处理后返回一个结果值，也就是数多个变量中的某一个值发生了变化则我们监控的这个值也就会发生变化，比如我们这里只要 currentUser 里面的任何一个值发生改变，这里就会同步更新，computed 和 vuex 的 state 可以说是”黄金搭配“了。

这样，我们就通过 vuex 完成了跨页面通用数据的共享和使用，这里的讲解有利于后续的开发，因为我们在后面会经常用到 vuex 来共享数据。

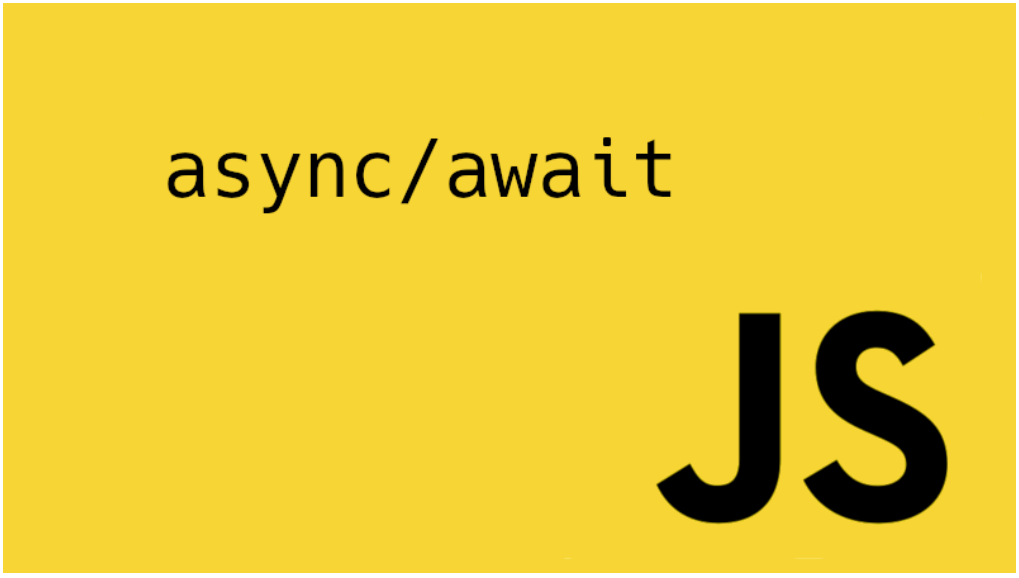
讲到这里，你可能会有一些问题，比如为什么请求数据的那部分逻辑不放在 vuex 里呢？这里笔者总结了2个使用 vuex 的原则和场景：

- vuex 本身是一个相对来说比较“绕”的一段逻辑，所以，并不是所有的场景都适合使用，当数据需要跨页面共享时，使用 vuex 是一个不错的选择。
- 像上面提到的登录校验这段，有请求返回的数据时，我们并没有将数据放在 vuex 里面，本身这块逻辑只和登录有关，这段请求数据的结果也只会在登录的页面用到（返回是否登录成功或者失败），数据并不具有共享性，所以没有必要故意去使用vuex。

async/await

async/await 语法在2016年就已经提出来了，属于 ES7 的其中一个测试标准（目前来看是直接跳过 ES7，列为了 ES8 的标准点这里查看），它主要为了解决下面两个问题：

1. 过多的嵌套回调问题。
2. 以 Promise 为主的链式回调问题。



如果说 Promise 解决了恐怖的嵌套回调问题，但是也引出了以 then 为主的链式回调问题，那么 async/await 就是拯救它们的救星。

顾名思义，async 关键字代表后面的函数中有异步操作，await表示等待一个异步方法执行完成。声明异步函数只需在普通函数前面加一个关键字async即可，代码如下：

```
async function myFunc() {}
```

async 函数返回一个 Promise 对象，因此 async 函数通过 return 返回的值，会成为 then 方法中回调函数的参数，代码如下：

```
async function myFunc() {
  return 'str!';
}

myFunc().then(value => {
  console.log(value); //str
})
```

但是，单独一个 async 函数，其实与 Promise 执行的功能是一样的，所以需要使用 await，代码如下：

```
let foo = await myFunc(); // str
```

await 表示等待一个 Promise，你可以把 await 命令理解成 then 命令的语法糖，值得注意的是， await 后面的 Promise 对象不总是返回 resolved 状态，只要一个 await 后面的Promise状态变为 rejected ，整个 async 函数都会中断执行，为了保存错误的位置和错误信息，我们需要用 try/catch 。

下面，以我们登录请求代码为例：

我们在 login.vue 里面有下面这段代码：

```
async signUp () {
  ...
  let resp = await service.post('users/signup', {
    phoneNum: this.phoneNum,
    code: this.code
  })
  ...
  this.$store.dispatch('setUser', resp.data)
},
```

由例子可以看出，`service.post()` 本身是一个异步请求，但是我们在方法前加上 `await` 之后，就可以把返回值赋值给变量 `resp`，然后之间在下一行写代码逻辑，可以理解 `await` 是核心，但是使用 `await` 有两个条件：

1. 使用 `await` 的代码必须被一个 `async` 修饰的方法包裹，正如例子中的 `signUp`，使用 `async` 修饰后就表明里面的代码是“非正常“逻辑一行一行运行的，而是含有异步操作的。
2. 基于 `await` 的方法，必须返回一个 Promise (这里的 service.post 返回的正好是一个 Promise)，它的返回结果是 Promise 的 `resolve` 结果，我们使用 `try/catch` 可以捕获 Promise 的 `rejected` 的结果。

```
try{
  ...
  let resp = await service.post('users/signup')
  ...
}catch(e){
  console.log(e)
}
```

掌握 `async/await` 的用法是非常重要的，因为后续我们在 Node.js 端也会大量使用。

小结

本小节主要讲解了 vuex 的基本概念和用法，以及 async/await 的原理和使用方法，为后续的代码做铺垫。
相关知识点：

1. vuex 主要用来做跨页面之间的组件通信，它帮助我们解决了页面之间的数据共享问题，但是它的逻辑通常会比较绕，所以并不是所有的数据都有必要通过vuex来管理。
2. async/await 是一个异步操作的结局方案，主要帮助解决 Promise 的链式回调问题，使用 async/await 来布局我们的代码结构，可以让代码逻辑更加清晰，便于维护，我们在后面的代码中会大量使用。

}

07 登录页 UI 界面开发

08 后端登录接口开发