

# Vue全家桶 & 原理

## 资源

1. [vue-router](#)
2. [vuex](#)
3. [vue-router源码](#)
4. [vuex源码](#)

## 知识点

### vue-router

Vue Router 是 [Vue.js](#) 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。

安装: `vue add router`

核心步骤：

- 步骤一：使用vue-router插件，router.js

```
import Router from 'vue-router'  
Vue.use(Router)
```

- 步骤二：创建Router实例，router.js

```
export default new Router({...})
```

- 步骤三：在根组件上添加该实例，main.js

```
import router from './router'  
new Vue({  
  router,  
}).$mount("#app");
```

- 步骤四：添加路由视图，App.vue

```
<router-view></router-view>
```

- 导航

```
<router-link to="/">Home</router-link>  
<router-link to="/about">About</router-link>
```

```
this.$router.push('/')  
this.$router.push('/about')
```

## vue-router源码实现

单页面应用程序中，url发生变化时候，不能刷新，显示对应视图内容

## 需求分析

- spa 页面不能刷新
  - hash #/about
  - History api /about
- 根据url显示对应的内容
  - router-view
  - 数据响应式：current变量持有url地址，一旦变化，动态重新执行render

## 任务

- 实现一个插件
  - 实现VueRouter类
    - 处理路由选项
    - 监控url变化，hashchange
    - 响应这个变化
  - 实现install方法
    - \$router注册
    - 两个全局组件

## 实现一个插件：创建VueRouter类和install方法

创建kvue-router.js

```

let Vue; // 引用构造函数，VueRouter中要使用

// 保存选项
class VueRouter {
  constructor(options) {
    this.$options = options;
  }
}

// 插件：实现install方法，注册$router
VueRouter.install = function(_Vue) {
  // 引用构造函数，VueRouter中要使用
  Vue = _Vue;

  // 任务1：挂载$router
  Vue.mixin({
    beforeCreate() {
      // 只有根组件拥有router选项
      if (this.$options.router) {
        // vm.$router
        Vue.prototype.$router = this.$options.router;
      }
    }
  });
}

// 任务2：实现两个全局组件router-link和router-view
Vue.component('router-link', Link)
Vue.component('router-view', View)
};

export default VueRouter;

```

为什么要用混入方式写？主要原因是use代码在前，Router实例创建在后，而install逻辑又需要用到该实例

## 创建router-view和router-link

创建krouter-link.js

```

export default {
  props: {
    to: String,
    required: true
  }
}

```

```

    },
    render(h) {
        // return <a href={'#' + this.to}>{this.$slots.default}</a>;
        return h('a', {
            attrs: {
                href: '#' + this.to
            }
        }, [
            this.$slots.default
        ])
    }
}

```

创建krouter-view.js

```

export default {
    render(h) {
        // 暂时先不渲染任何内容
        return h(null);
    }
}

```

## 监控url变化

定义响应式的current属性，监听hashchange事件

```

class VueRouter {
    constructor(options) {
        // 定义响应式的属性current
        const initial = window.location.hash.slice(1) || '/'
        Vue.util.defineReactive(this, 'current', initial)

        // 监听hashchange事件
        window.addEventListener('hashchange', this.onHashChange.bind(this))
        window.addEventListener('load', this.onHashChange.bind(this))
    }

    onHashChange() {
        this.current = window.location.hash.slice(1)
    }
}

```

动态获取对应组件，krouter-view.js

```

export default {
  render(h) {
    // 动态获取对应组件
    let component = null;
    const route = this.$router.$options.routes.find(route => route.path ===
this.$router.current)
    if(route) component = route.component
    return h(component);
  }
}

```

## 提前处理路由表

提前处理路由表避免每次都循环

```

class VueRouter {
  constructor(options) {
    // 缓存path和route映射关系
    this.routeMap = {}
    this.$options.routes.forEach(route => {
      this.routeMap[route.path] = route
    });
  }
}

```

使用，krouter-view.js

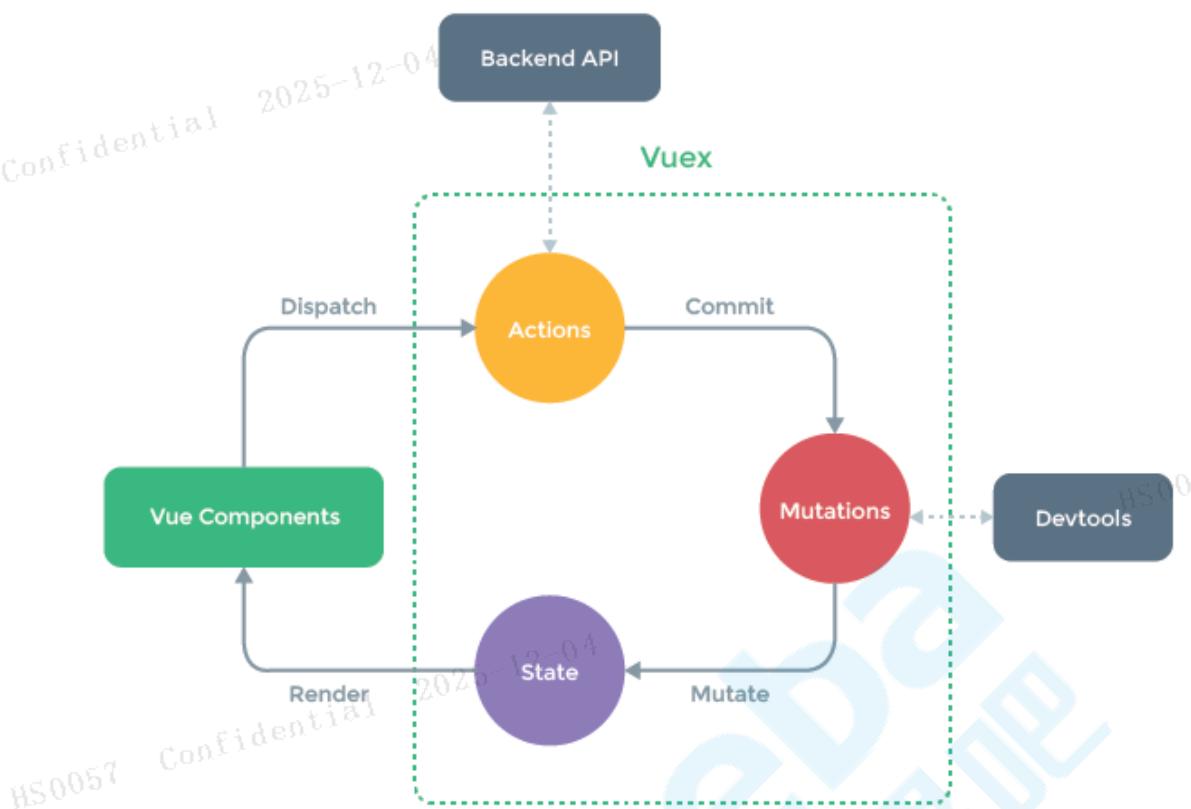
```

export default {
  render(h) {
    const {routeMap, current} = this.$router
    const component = routeMap[current] ? routeMap[current].component : null;
    return h(component);
  }
}

```

## Vuex

Vuex 集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以可预测的方式发生变化。



## 整合vuex

```
vue add vuex
```

## 核心概念

- state 状态、数据
- mutations 更改状态的函数
- actions 异步操作
- store 包含以上概念的容器

## 状态 - state

state保存应用状态

```
export default new Vuex.Store({
  state: { counter: 0 },
})
```

## 状态变更 - mutations

mutations用于修改状态，store.js

```
export default new Vuex.Store({
  mutations: {
    add(state) {
      state.counter++
    }
  }
})
```

## 派生状态 - getters

从state派生出新状态，类似计算属性

```
export default new Vuex.Store({
  getters: {
    doubleCounter(state) { // 计算剩余数量
      return state.counter * 2;
    }
  }
})
```

## 动作 - actions

添加业务逻辑，类似于controller

```
export default new Vuex.Store({
  actions: {
    add({ commit }) {
      setTimeout(() => {
        commit('add')
      }, 1000);
    }
  }
})
```

测试代码：

```
<p @click="$store.commit('add')">counter: {{$store.state.counter}}</p>
<p @click="$store.dispatch('add')">async counter: {{$store.state.counter}}</p>
<p>double: {{$store.getters.doubleCounter}}</p>
```

# vux原理解析

## 任务分析

- 实现插件
  - 实现Store类
    - 维持一个响应式状态state
    - 实现commit()
    - 实现dispatch()
    - getters
  - 挂载\$store

初始化：Store声明、install实现，k vuex.js：

```
let _Vue;

class Store {
  constructor(options = {}) {
    this._vm = new Vue({
      data: {
        $$state:options.state
      }
    });
    get state() {
      return this._vm._data.$$state
    }

    set state(v) {
      console.error('please use replaceState to reset state');
    }
  }
}

function install(_Vue) {
  _Vue = _Vue;

  _Vue.mixin({
    beforeCreate() {
      if (_this.$options.store) {
        _Vue.prototype.$store = _this.$options.store;
      }
    }
  });
}
```

```
export default { Store, install };
```

实现commit: 根据用户传入type获取并执行对应mutation

```
class Store {
  constructor(options = {}) {
    // 保存用户配置的mutations选项
    this._mutations = options.mutations || {}
  }

  commit(type, payload) {
    // 获取type对应的mutation
    const entry = this._mutations[type]

    if (!entry) {
      console.error(`unknown mutation type: ${type}`);
      return
    }
    // 指定上下文为Store实例
    // 传递state给mutation
    entry(this.state, payload);
  }
}
```

实现actions: 根据用户传入type获取并执行对应action

```
class Store {
  constructor(options = {}) {
    // 保存用户编写的actions选项
    this._actions = options.actions || {}

    // 绑定commit上下文否则action中调用commit时可能出问题!
    // 同时也把action绑了, 因为action可以互调
    const store = this
    const {commit, action} = store
    this.commit = function boundCommit(type, payload) {
      commit.call(store, type, payload)
    }
    this.action = function boundAction(type, payload) {
      return action.call(store, type, payload)
    }
  }

  dispatch(type, payload) {
    // 获取用户编写的type对应的action
    const entry = this._actions[type]
```

```
if (!entry) {  
    console.error(`unknown action type: ${type}`);  
    return  
}  
// 异步结果处理常常需要返回Promise  
return entry(this, payload);  
}  
}
```

## 作业

在kvuex中实现getters

## 思考拓展

1. 尝试去看看vue-router的[源码](#)，并解决嵌套路由的问题
2. 提前了解vue数据响应原理为下节课做准备