勘误

```
watch: {
    $route: {
        handler(route) {
            console.log(this.$route.matched);
            this.crumbData = this.$route.matched.map(m => m.name ||
m.redirect);
        },
        immediate: true // 这一行要加上，让它一开始执行一次
    }
},
```

复习

弹窗类组件

递归组件

vue-router应用

vue-router原理

拓展

vue插件

```
// 插件定义
MyPlugin.install = function (Vue, options) {
  // 1. 添加全局方法或属性
  Vue.myGlobalMethod = function () {
    // 逻辑...
  }

  // 2. 添加全局资源
  Vue.directive('my-directive', {
    bind (el, binding, vnode, oldVnode) {
      // 逻辑...
    }
    ...
  })

  // 3. 注入组件选项
```

```
  Vue.mixin({
    created: function () {
      // 逻辑...
    }
    ...
  })

  // 4. 添加实例方法
  Vue.prototype.$myMethod = function (methodOptions) {
    // 逻辑...
  }
}

// 插件使用
Vue.use(MyPlugin)
```

范例：移动$bus到插件

组件混入：mixin

混入 (mixin) 提供了一种分发 Vue 组件中可复用功能的灵活方式

```
// 定义一个混入对象
var myMixin = {

  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// 定义一个使用混入对象的组件
var Component = Vue.extend({
  mixins: [myMixin]
})
```

范例：移动dispatch和broadcast到mixins

render函数详解

一些场景中需要 JavaScript 的完全编程的能力，这时可以用**渲染函数**，它比模板更接近编译器。

render(h) {

return h(tag, {...}, [children])

}

createElement函数

```
{
  // 与 `v-bind:class` 的 API 相同,
  // 接受一个字符串、对象或字符串和对象组成的数组
  'class': {
    foo: true,
    bar: false
  },
  // 与 `v-bind:style` 的 API 相同,
  // 接受一个字符串、对象，或对象组成的数组
  style: {
    color: 'red',
    fontSize: '14px'
  },
  // 普通的 HTML 特性
  attrs: {
    id: 'foo'
  },
  // 组件 prop
  props: {
    myProp: 'bar'
  },
  // DOM 属性
  domProps: {
    innerHTML: 'baz'
  },
  // 事件监听器在 `on` 属性内,
  // 但不再支持如 `v-on:keyup.enter` 这样的修饰器。
  // 需要在处理函数中手动检查 keyCode。
  on: {
    click: this.clickHandler
  },
}
```

作业

尝试去看看vue-router的源码，并回答router-view嵌套的问题是如何解决的

函数式组件

组件若没有管理任何状态，也没有监听任何传递给它的状态，也没有生命周期方法，只是一个接受一些 prop 的，可标记为函数式组件，此时它没有上下文

> - `props`：提供所有 prop 的对象
> - `children`: VNode 子节点的数组
> - `slots`: 一个函数，返回了包含所有插槽的对象
> - `scopedSlots`: (2.6.0+) 一个暴露传入的作用域插槽的对象。也以函数形式暴露普通插槽。
> - `data`：传递给组件的整个数据对象，作为 `createElement` 的第二个参数传入组件
> - `parent`：对父组件的引用
> - `listeners`: (2.3.0+) 一个包含了所有父组件为当前组件注册的事件监听器的对象。这是 `data.on` 的一个别名。
> - `injections`: (2.3.0+) 如果使用了 `inject` 选项，则该对象包含了应当被注入的属性。

文档

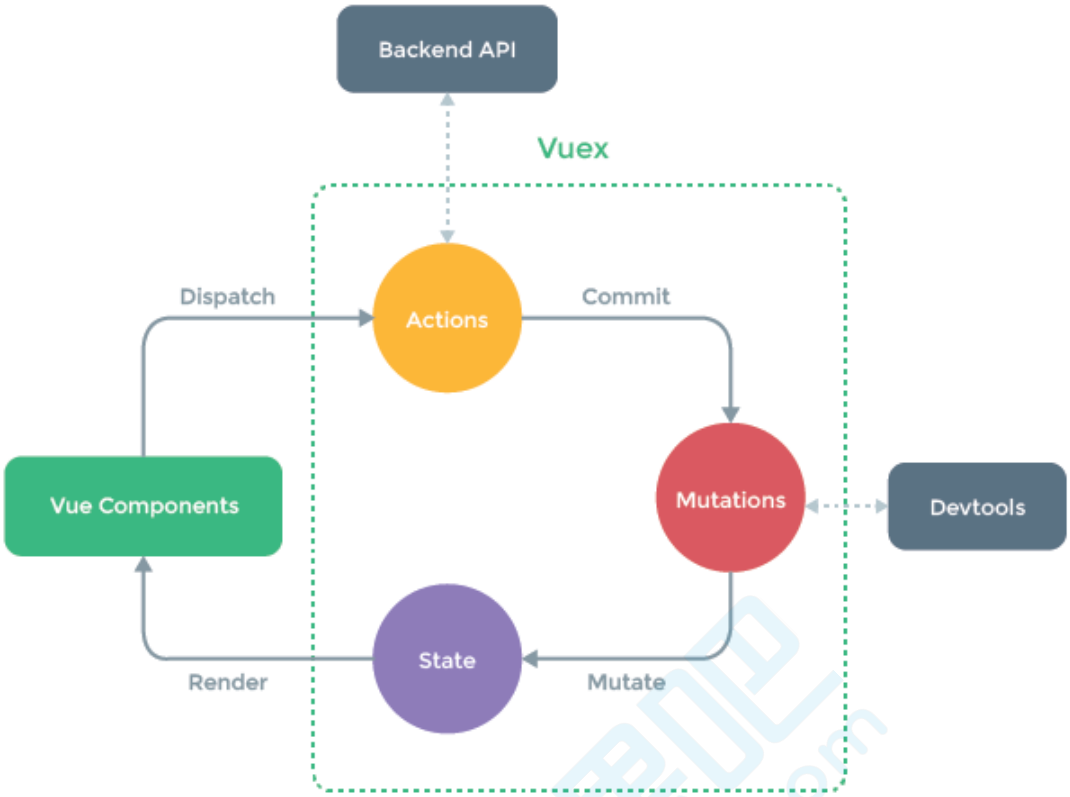
知识点

Vuex数据管理

Vuex 是一个专为 Vue.js 应用开发的**状态管理模式**，集中式存储管理应用所有组件的状态。

Vuex遵循“单向数据流”理念，易于问题追踪以及提高代码可维护性。

Vue中多个视图依赖于同一状态时，视图间传参和状态同步比较困难，Vuex能够很好解决该问题。



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

状态和状态变更

state保存数据状态，mutations用于修改状态，store.js

```
export default new Vuex.Store({
  state: { count:0 },
  mutations: {
    increment(state, n = 1) {
      state.count += n;
    }
  }
})
```

使用状态，vuex/index.vue

```
<template>
  <div>
    <div>冲啊，手榴弹扔了{{$store.state.count}}个</div>
    <button @click="add">扔一个</button>
  </div>
</template>

<script>
export default {
  methods: {
    add() {
      this.$store.commit("increment");
    }
  }
};
</script>
```

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: {
    score(state) {
      return `共扔出：${state.count}`
    }
  }
})
```

登录状态文字，App.vue

```
<span>{{$store.getters.score}}</span>
```

动作 - actions

复杂业务逻辑，类似于controller

```
export default new Vuex.Store({
  actions: {
    incrementAsync({ commit }) {
      setTimeout(() => {
        commit("increment", 2);
      }, 1000);
    }
  }
})
```

使用actions：

```
<template>
  <div id="app">
    <div>冲啊，手榴弹扔了{{$store.state.count}}个</div>
    <button @click="addAsync">蓄力扔俩</button>
  </div>
</template>

<script>
export default {
  methods: {
    addAsync() {
      this.$store.dispatch("incrementAsync");
    }
};
</script>
```

模块化

按模块化的方式编写代码，store.js

```
const count = {
  namespaced: true,
  // ...
};

export default new Vuex.Store({
    modules: {a: count}
});
```

使用变化，components/vuex/module.vue

```
<template>
  <div id="app">
    <div>冲啊，手榴弹扔了{{$store.state.a.count}}个</div>
    <p>{{$store.getters['a/score']}}</p>
    <button @click="add">扔一个</button>
    <button @click="addAsync">蓄力扔俩</button>
  </div>
</template>

<script>
export default {
  methods: {
    add() {
      this.$store.commit("a/increment");
    },
    addAsync() {
      this.$store.dispatch("a/incrementAsync");
    }
  }
};
</script>
```

vuex原理解析

初始化：Store声明、install实现，， kvuex.js：

```
let Vue;

function install(_Vue) {
  Vue = _Vue;

  // 这样store执行的时候，就有了Vue，不用import
  // 这也是为啥Vue.use必须在新建store之前
  Vue.mixin({
    beforeCreate() {
      // 这样才能获取到传递进来的store
      // 只有root元素才有store，所以判断一下
      if (this.$options.store) {
        Vue.prototype.$store = this.$options.store;
      }
    }
  });
}

class Store {
```

```
  constructor(options = {}) {
    this.state = new Vue({
      data: options.state
    });
    this.mutations = options.mutations || {};
  }
  // 注意这里用箭头函数形式，后面actions实现时会有作用
  commit = (type, arg) => {
    this.mutations[type](this.state, arg);
  };

}

export default { Store, install };
```

实现actions

```
class Store {
  constructor(options = {}) {
    this.actions = options.actions;
  }

  dispatch(type, arg) {
    this.actions[type](
      {
        commit: this.commit,
        state: this.state
      },
      arg
    );
  }

}
```

实现getters

```
class Store {
  constructor(options = {}) {
    options.getters && this.handleGetters(options.getters);
  }

  handleGetters(getters) {
    this.getters = {}; // 定义this.getters
    // 遍历getters选项，为this.getters定义property
    // 属性名就是选项中的key，只需定义get函数保证其只读性
    Object.keys(getters).forEach(key => {
      // 这样这些属性都是只读的
      Object.defineProperty(this.getters, key, {
```

```
        get: () => { // 注意依然是箭头函数
          return getters[key](this.state);
        }
      });
    });
  }
}
```

作业

1. 复习消化前两节内容
2. 了解vue原理为下节课做准备
3. 简版vue：数据响应化、编译器 Object.defineProperty