Vue全家桶 & 原理

勘误

1. 兄弟组件通信借用$parent的例子有误

```
mounted() {
    // 通过共同父组件做中介
    this.$parent.$on("foo", comp => {
        if (comp !== this) {
            console.log("foo~~");
        }
    });
},
methods: {
    saySth2Bro() {
        // 父组件派发消息可以在所有兄弟内监听
        this.$parent.$emit("foo", this);
    }
}
```

2. 笔记中FormItem执行校验代码少括号

```
mounted () {
    this.$on('validate', () => {
        this.validate()
    });
}
```

作业

1. 尝试解决Input里面$parent派发事件不够健壮的问题

   element的minxins方法

   Input组件中的使用

2. 说出.sync和v-model的异同

```
<template>
    <!--v-model是语法糖-->
    <Input v-model="username">
    <!--默认等效于下面这行-->
    <Input :value="username" @input="username=$event">
</template>

<script>
// 但是你也可以通过设置model选项修改默认行为，Checkbox.vue
{
```

```
    model: {
        prop: 'checked',
        event: 'change'
    }
}
</script>

<template>
    <!--上面这样设置会导致上级使用v-model时行为变化，相当于-->
    <KCheckBox :checked="model.remember"
               @change="model.remember = $event"></KCheckBox>
</template>

// v-model通常用于表单控件，这样子组件有更强控制能力。

<template>
    <!-- sync修饰符添加于v2.4，它能用于修改传递到子组件的属性，如果像下面这样写  -->
    <Input :value.sync="model.username">
    <!-- 等效于下面这行，那么和v-model的区别只有事件名称的变化 -->
    <Input :foo="username" @update:foo="username=$event">
</template>

<input @input="$emit('update:foo', $event)"/>

<!-- 这里绑定属性名称更改，相应的属性名也会变化 -->
<Input :foo="username" @update:foo="username=$event">

// 场景：父组件传递的属性子组件想修改
// 所以sync修饰符的控制能力都在父级，事件名称也相对固定update:xx
// 习惯上表单元素用v-model
```

资源

1. vue-router
2. vuex
3. vue-router源码
4. vuex源码

知识点

vue-router

安装：

```
vue add router
```

起步

配置，router.js

```
{ path: '/', name: 'home', component: Home }
```

路由出口

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

导航链接

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

添加到vue选项

```
// main.js
new Vue({
  router, // 为什么挂载
  render: h => h(App)
}).$mount('#app')
```

动态路由

把某种模式匹配到的所有路由，全都映射到同一个组件。

~ 创建Detail.vue

```
<template>
    <div>
        <h2>商品详情</h2>
        <p>{{$route.params.id}}</p>
    </div>
</template>
```

~ 详情页路由配置，router.js

```
{
    path: '/detail/:id',
    name: 'detail',
    component: Detail
}
```

~ 跳转，Home.vue

```
<div v-for="item in items" :key="item.id">
    <router-link :to="`/detail/${item.id}`">
        {{ item.name }}
    </router-link>
</div>
```

路由嵌套

```
{
    path: '/',
    name: 'home',
    component: Home,
    children: [{
        path: '/detail/:id',
        name: 'detail',
        component: Detail
    }]
}
```

~ 添加插座，Home.vue

```
<template>
    <div class="home">
        <h1>首页</h1>
        <router-view></router-view>
    </div>
</template>
```

vue-router源码实现

VueRouter基本结构，创建kvue-router.js

```
class VueRouter {
  constructor(options) {
    this.$options = options;
    // path、component映射
    this.routeMap = {};
    // current保存当前hash
    // vue使其是响应式的
    this.app = new Vue({
      data: {
        current: "/"
      }
    });
  }
}

let Vue;

// 插件逻辑
VueRouter.install = function(_Vue) {
  Vue = _Vue;

  Vue.mixin({
    beforeCreate() {
      if (this.$options.router) {
        // 确保是根组件时执行一次，将router实例放到Vue原型，以后所有组件实例就均有$router
        Vue.prototype.$router = this.$options.router;
        this.$options.router.init();
      }
    }
  });
```

```
};
```

编写VueRouter初始化逻辑

```
class VueRouter {
  // ...

  init() {
    this.bindEvents();
    this.createRouteMap();
    this.initComponent();
  }
  // hash变更检测
  bindEvents() {
    window.addEventListener("load", this.onHashChange.bind(this), false);
    window.addEventListener("hashchange", this.onHashChange.bind(this), false);
  }

  // 路径变更处理
  onHashChange() {
    this.app.current = window.location.hash.slice(1) || "/";
  }

  // 创建路由映射表
  createRouteMap() {
    this.$options.routes.forEach(item => {
      this.routeMap[item.path] = item;
    });
  }

  // router-link/router-view声明
  initComponent() {
    Vue.component("router-link", {
      props: {
        to: String
      },
      render(h) {
        return <a href={this.to}>{this.$slots.default}</a>;
        // return h('a',{
        //     attrs:{
        //         href:'#'+this.to
        //     }
        // },[
        //     this.$slots.default
        // ])
      }
    });
    Vue.component("router-view", {
      render: h => {
        var component = this.routeMap[this.app.current].component;
        return h(component);
      }
    });
  }
}
```

使用自定义路由，创建路由配置文件：krouter.js

```
import Home from "./views/Home";
import About from "./views/About";

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
      { path: "/", component: Home },
      { path: "/about", component: About }
  ]
});
```

引入路由，main.js

```
import router from './krouter'
```

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。

整合vuex

```
vue add vuex
```

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) {
      state.count += 1;
    }
  }
})
```

使用状态，Home.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: {
    left(state) { // 计算剩余数量
      return 10 - state.count;
    }
  }
})
```

登录状态文字，App.vue

```
<span>还剩{{$store.getters.left}}个</span>
```

动作 - actions

复杂业务逻辑，类似于controller

```
export default new Vuex.Store({
  actions: {
    increment({ getters, commit }) {
      // 添加业务逻辑
      if (getters.left > 0) {
        commit("increment");
        return true;// 返回结果
      }
      return false;// 返回结果
    },
    asyncIncrement({ dispatch }) {
      // 异步逻辑返回Promise
      return new Promise(resolve => {
        setTimeout(() => {
          // 复用其他action
          resolve(dispatch("increment"));
        }, 1000);
      });
    },
  }
})
```

使用actions：

```
<template>
  <div id="app">
    <button @click="asyncAdd">蓄力扔一个</button>
  </div>
</template>

<script>
export default {
  methods: {
    add() {
      // 即使action执行同步代码返回的结果依然是promise
      this.$store.dispatch("increment").then(result => {
        if (!result) {
          alert("投掷失败！没货啦");
        }
      });
      // this.$store.commit("increment");
    },
    asyncAdd() {
      this.$store.dispatch("asyncIncrement").then(result => {
        if (!result) {
          alert("投掷失败！没货啦");
        }
      });
    }
};
</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
    );
  }
}
```

作业

1. 尝试去看看vue-router的源码，并解答：嵌套路由的解决方式
2. 完成kvuex中getters部分

预告

手写vue，着重实现数据响应化、编译。