复习

1.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.v-model和.sync

```
<!--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">

// 但是你也可以通过设置model选项修改默认行为，Input.vue
{
    model: {
        prop: 'checked',
        event: 'change'
    }
}
// 上面这样设置会导致上级使用v-model时行为变化，相当于
// v-model通常用于表单控件，它有默认行为，属性名和事件名均可定义
<Input :checked="username" @change="username=$event">


<!-- sync修饰符类似于v-model，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="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 :value="username" @update:value="username=$event">
<!-- 这里绑定属性名称可以随意更改，相应的属性名也会变化 -->
<Input :foo="username" @update:foo="username=$event">
// 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
```

create再串一下

```
// Component vs comp
// Component: 组件配置，js对象
// comp：组件实例
// Vue.extend(Component) => function 组件构造函数
// Vue.component('comp', Component 全局注册组件
// create把传递的组件配置转换为组件实例返回
function create(Component, props) {
    // 先创建vue实例，用它创建组件实例
    const vm = new Vue({
        render(h) {
            // h就是createElement，它返回VNode
            return h(Component, {props})
        }
    }).$mount(); // $mount里面会调render生成VNode，生成的VNode会执行update函数生成DOM

    // 手动挂载：生成DOM结构存储在vm.$el把它追加到body即可
```

```
    document.body.appendChild(vm.$el);

    // 从vm.$children中拿出comp
    const comp = vm.$children[0];// vm.$root也是comp
    // 销毁方法
    comp.remove = function() {
        document.body.removeChild(vm.$el);
        vm.$destroy();
    }
    return comp;
}
```

掌握vue-router用法和技巧

配置

```
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home
    },
    {
      path: '/about',
      name: 'about',
      // 路由层级代码分割，生成分片(about.[hash].js)
      // 当路由房问时会懒加载.
      component: () => import(/* webpackChunkName: "about" */
'./views/About.vue')
    }
  ]
```

指定路由器

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

路由视图

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

导航链接

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

路由嵌套

```
// router.js
{
    path: "/",
    component: Home,
    children: [{ path: "/list", name: "list", component: List }]
}

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

动态路由

```
{ path: "detail/:id", component: Detail },

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

理解vue-router实现原理

```
class VueRouter {
  constructor(options) {
    this.$options = options;
    this.routeMap = {};

    // 路由响应式
    this.app = new Vue({
      data: {
        current: "/"
      }
    });
  }

  init() {
    this.bindEvents(); //监听url变化
    this.createRouteMap(this.$options); //解析路由配置
    this.initComponent(); // 实现两个组件
  }

  bindEvents() {
    window.addEventListener("load", this.onHashChange.bind(this));
    window.addEventListener("hashchange", this.onHashChange.bind(this));
  }
  onHashChange() {
```

```
    this.app.current = window.location.hash.slice(1) || "/";
  }
  createRouteMap(options) {
    options.routes.forEach(item => {
      this.routeMap[item.path] = item.component;
    });
  }
  initComponent() {
    // router-link,router-view
    // <router-link to="">fff</router-link>
    Vue.component("router-link", {
      props: { to: String },
      render(h) {
        // h(tag, data, children)
        return h("a", { attrs: { href: "#" + this.to } }, [
          this.$slots.default
        ]);
      }
    });

    // <router-view></router-view>
    Vue.component("router-view", {
      render: h => {
        const comp = this.routeMap[this.app.current];
        return h(comp);
      }
    });
  }
}
VueRouter.install = function(Vue) {
  // 混入
  Vue.mixin({
    beforeCreate() {
      // this是Vue实例
      if (this.$options.router) {
        // 仅在根组件执行一次
        Vue.prototype.$router = this.$options.router;
        this.$options.router.init();
      }
    }
  });
};
```

掌握vuex理念和核心用法

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：

```
this.$store.dispatch("increment").then(result => {
    if (!result) {
        alert("投掷失败！没货啦");
    }
});
```

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

```
<div>冲啊，手榴弹扔了{{$store.state.a.count}}个</div>
<p>{{$store.getters['a/score']}}</p>

this.$store.commit("a/increment");
this.$store.dispatch("a/incrementAsync");
```

vuex原理解析

```
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 || {};
      this.actions = options.actions;
       options.getters && this.handleGetters(options.getters);
  }
  // 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);
  };
dispatch(type, arg) {
    this.actions[type](
      {
        commit: this.commit,
        state: this.state
      },
      arg
    );
  }
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);
        }
      });
    });
  }
}

export default { Store, install };
```