复习

作业

1. 能手写Form、FormItem、Input实现
2. 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
3. 说出.sync和v-model的异同

```
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，Checkbox.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
<KCheckBox :checked="model.remember" @change="model.remember = $event">
</KCheckBox>

// 场景：v-model通常用于表单控件，它有默认行为，同时属性名和事件名均可在子组件定义


<!-- sync修饰符添加于v2.4，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="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 :value="username" @update:value="username=$event">
<!-- 这里绑定属性名称更改，相应的属性名也会变化 -->
<Input :foo="username" @update:foo="username=$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
```

主要内容：

1. 弹窗类组件设计和实现
2. tree组件实现
3. 路由vue-router
4. vue-router实现原理

实现弹窗组件

弹窗这类组件的特点是它们在当前vue实例之外独立存在，通常挂载于body；它们是通过JS动态创建的，不需要在任何组件中声明。常见使用姿势：

```
this.$create(Notice, {
    title: '社会你杨哥喊你来搬砖',
    message: '提示信息',
    duration: 1000
}).show();
```

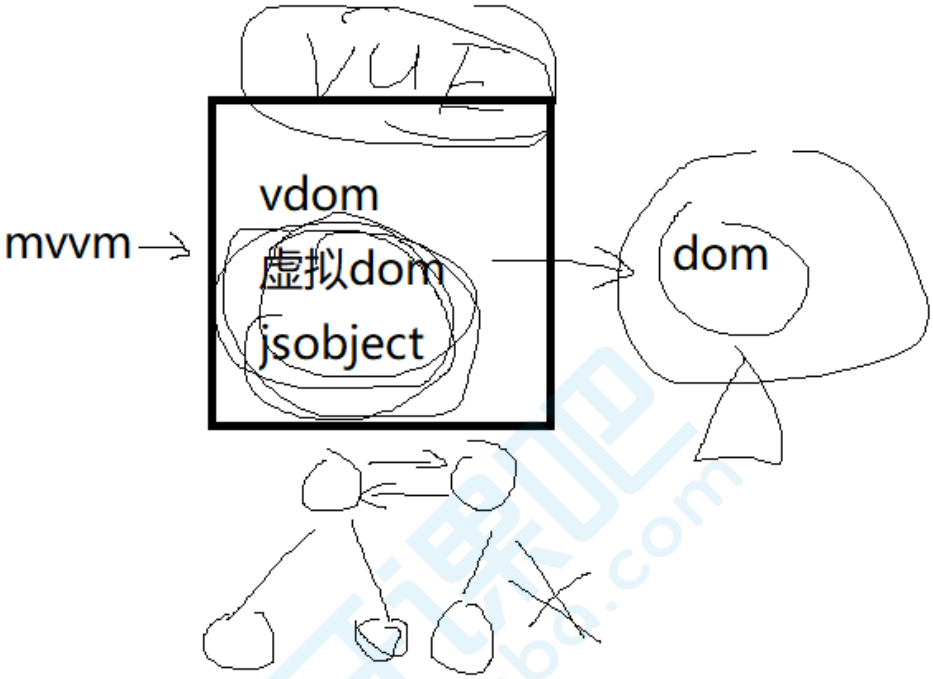
create

create函数用于动态创建指定组件实例并挂载至body

```
import Vue from "vue";

// 创建函数接收要创建组件定义
function create(Component, props) {
  // 创建一个Vue新实例
  const vm = new Vue({
    render(h) {
      // render函数将传入组件配置对象转换为虚拟dom
      console.log(h(Component, { props }));
      return h(Component, { props });
    }
  }).$mount(); //执行挂载函数，但未指定挂载目标，表示只执行初始化工作

  // 将生成dom元素追加至body
  document.body.appendChild(vm.$el);

  // 给组件实例添加销毁方法
  const comp = vm.$children[0];
  comp.remove = () => {
    document.body.removeChild(vm.$el);
    vm.$destroy();
  };
```

```
  return comp;
}

// 暴露调用接口
export default create;
```

> render函数的作用是得到描述dom结构的虚拟dom



创建通知组件，Notice.vue

```
<template>
  <div class="box" v-if="isShow">
    <h3>{{title}}</h3>
    <p class="box-content">{{message}}</p>
  </div>
</template>

<script>
export default {
  props: {
    title: {
      type: String,
      default: ""
    },
    message: {
      type: String,
      default: ""
    },
    duration: {
      type: Number,
```

```
      default: 1000
    }
  },
  data() {
    return {
      isShow: false
    };
  },
  methods: {
    show() {
      this.isShow = true;
      setTimeout(this.hide, this.duration);
    },
    hide() {
      this.isShow = false;
      this.remove();
    }
  }
};
</script>

<style>
.box {
  position: fixed;
  width: 100%;
  top: 16px;
  left: 0;
  text-align: center;
  pointer-events: none;
}
.box-content {
  width: 200px;
  margin: 10px auto;
  font-size: 14px;
  border: blue 3px solid;
  padding: 8px 16px;
  background: #fff;
  border-radius: 3px;
  margin-bottom: 8px;
}
</style>
```

使用create api

测试，components/form/index.vue

```
<script>
import create from "@/utils/create";
import Notice from "@/components/Notice";
```

```
export default {
  methods: {
    submitForm(form) {
      this.$refs[form].validate(valid => {
        const notice = create(Notice, {
          title: "社会你杨哥喊你来搬砖",
          message: valid ? "请求登录!" : "校验失败!",
          duration: 1000
        });
        notice.show();
      });
    }
  }
};
</script>
```

递归组件

递归组件是可以在它们自己模板中调用自身的组件。

```
// Node.vue
<template>
    <div>
        <h3>{{data.title}}</h3>
        <!-- 必须有结束条件 -->
        <Node v-for="d in data.children" :key="d.id" :data="d"></Node>
    </div>
</template>

<script>
    export default {
        name: 'Node', // name对递归组件是必要的
        props: {
            data: {
                type: Object,
                require: true
            },
        },
    }
</script>

// 使用
<Node :data="{id:'1',title:'递归组件',children:[{...}]}"></Node>
```

实现Tree组件

Tree组件是典型的递归组件，其他的诸如菜单组件都属于这一类，也是相当常见的。

组件设计

Tree组件最适合的结构是无序列表ul，创建一个递归组件Item表示Tree选项，如果当前Item存在children，则递归渲染子树，以此类推；同时添加一个标识管理当前层级item的展开状态。

实现Item组件

```
<template>
  <li>
    <div @click="toggle">
      {{model.title}}
      <span v-if="isFolder">[{{open ? '-' : '+'}}]</span>
    </div>
    <ul v-show="open" v-if="isFolder">
      <item class="item" v-for="model in model.children" :model="model"
:key="model.title"></item>
    </ul>
  </li>
</template>

<script>
export default {
  name: "Item",
  props: {
    model: Object
  },
  data: function() {
    return {
      open: false
    };
  },
  computed: {
    isFolder: function() {
      return this.model.children && this.model.children.length;
    }
  },
  methods: {
    toggle: function() {
      if (this.isFolder) {
        this.open = !this.open;
      }
    },
  }
```

```
};
</script>
```

使用

```
<template>
  <div id="app">
    <ul>
      <item class="item" :model="treeData"></item>
    </ul>
  </div>
</template>
<script>
import Item from "./components/Item";
export default {
  name: "app",
  data() {
    return {
      treeData: {
        title: "Web全栈架构师",
        children: [
          {
            title: "Java架构师"
          },
          {
            title: "JS高级",
            children: [
              {
                title: "ES6"
              },
              {
                title: "动效"
              }
            ]
          },
          {
            title: "Web全栈",
            children: [
              {
                title: "Vue训练营",
                expand: true,
                children: [
                  {
                    title: "组件化"
                  },
                  {
                    title: "源码"
                  },
                  {
```

```
                    title: "docker部署"
                  }
                ]
              },
              {
                title: "React",
                children: [
                  {
                    title: "JSX"
                  },
                  {
                    title: "虚拟DOM"
                  }
                ]
              },
              {
                title: "Node"
              }
            ]
          }
        ]
      }
    };
  },
  components: { Item }
};
</script>
```

vue-router

安装:

```
vue add router
```

起步

配置

```
// router.js
import Vue from 'vue'
import Router from 'vue-router'
import Home from './views/Home.vue'
```

```
Vue.use(Router) // 引入Router插件

export default new Router({
  mode: 'history', // 模式: hash | history | abstract
  base: process.env.BASE_URL, // http://localhost:8080/cart
  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

应用界面通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件。

配置嵌套路由，router.js

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

父组件需要添加插座，Home.vue

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

动态路由

我们经常需要把某种模式匹配到的所有路由，全都映射到同一个组件。

详情页路由配置，router.js

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

跳转，List.vue

```
<ul>
    <li><router-link to="/detail/1">web全栈</router-link></li>
</ul>
```

获取参数，Detail.vue

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

> 传递路由组件参数：
>
> ```
> { path: "detail/:id", component: Detail, props: true }
> ```
>
> 组件中以属性方式获取：
>
> ```
> export default { props: ['id'] }
> ```

路由守卫

路由导航过程中有若干生命周期钩子，可以在这里实现逻辑控制。

全局守卫，router.js

```
// 路由配置
{
    path: "/about",
    name: "about",
    meta: { auth: true }, // 需要认证
    component: () => import(/* webpackChunkName: "about" */
"./views/About.vue")
}

// 守卫
router.beforeEach((to, from, next) => {
  // 要访问/about且未登录需要去登录
  if (to.meta.auth && !window.isLogin) {
    if (window.confirm("请登录")) {
      window.isLogin = true;
      next();   // 登录成功，继续
    } else {
      next('/');// 放弃登录，回首页
    }
  } else {
    next(); // 不需登录，继续
  }
});
```

路由独享守卫

```
beforeEnter(to, from, next) {
    // 路由内部知道自己需要认证
    if (!window.isLogin) {
        // ...
    } else {
        next();
    }
},
```

组件内的守卫

```
export default {
  beforeRouteEnter(to, from, next) {},
  beforeRouteUpdate(to, from, next) {},
  beforeRouteLeave(to, from, next) {}
};
```

vue-router拓展

动态路由

利用$router.addRoutes()可以实现动态路由添加，常用于用户权限控制。

```
// router.js
// 返回数据可能是这样的
//[{
//  path: "/",
//  name: "home",
//  component: "Home", //Home
//}]

// 异步获取路由
api.getRoutes().then(routes => {
    const routeConfig = routes.map(route => mapComponent(route));
    router.addRoutes(routeConfig);
})

// 映射关系
const compMap = {
  'Home': () => import("./view/Home.vue")
}

// 递归替换
function mapComponent(route) {
```

```
    route.component = compMap[route.component];
    if(route.children) {
        route.children = route.children.map(child => mapComponent(child))
    }
    return route
}
```

面包屑

利用$route.matched可得到路由匹配数组，按顺序解析可得路由层次关系。

```
// Breadcrumb.vue
watch: {
    $route() {
        // [{name:'home',path:'/'},{name:'list',path:'/list'}]
        console.log(this.$route.matched);
        // ['home','list']
        this.crumbData = this.$route.matched.map(m => m.name)
    }
}
```

vue-router源码实现

通常用法

```
// krouter.js
import Home from "./views/Home";
import About from "./views/About";
import VueRouter from "./kvue-router";

Vue.use(VueRouter);

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

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

具体实现

```
// kvue-router.js
let Vue;

class VueRouter {
  constructor(options) {
    this.$options = options;
    this.routeMap = {};
    this.app = new Vue({
      data: {
        current: "/"
      }
    });
  }
  // 绑定事件
  init() {
    this.bindEvents();
    this.createRouteMap(this.$options);
    this.initComponent();
  }
  bindEvents() {
    window.addEventListener("load", this.onHashChange.bind(this), false);
    window.addEventListener("hashchange", this.onHashChange.bind(this),
false);
  }
  // 路由映射表
  createRouteMap(options) {
    options.routes.forEach(item => {
      this.routeMap[item.path] = item;
    });
  }

  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);
      }
    });
  }

  // 设置当前路径
  onHashChange() {
    this.app.current = window.location.hash.slice(1) || "/";
  }
}
// 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();
      }
    }
  });
};
```

> 测试注意：
>
> 不要出现router-view嵌套，因为没有考虑，把Home中的router-view先禁用
>
> 导航链接修改为hash

作业

尝试去看看vue-router的源码，并回答如何解决router-view嵌套的情形