vue-router学习

1. 路由的简单配置

- html代码

```
<div id="app">
    <!--使用router-link组件来导航-->
    <router-link to='/home'>home</router-link>
    <router-link to='/product'>product</router-link>
    <!--路由出口-->
    <!--路由匹配到的组件将渲染在这里-->
    <router-view></router-view>
</div>
```

- js代码

```
<script>
    // vue-rotuer:
    //    分为3种：1.普通路由  2.动态路由（传参） 3.嵌套路由

    // 1.定义路由跳转的组件,可以从其它地方import进来
    var home = {template:"<h1>我是home</h1>"};
    var product = {template:"<h2>我是product</h2>"};
    // 2.实例化路由
    var router = new VueRouter({
        // 定义路由（普通路由）
        routes:[
            {name:'home',path:'/home',component:home},
            {name:'product',path:'/product',component:product}
        ]
    });
    // 3.创建和挂载根实例
    var vm = new Vue({
        el:"#app",
        data:function () {
            return {

            }
        },
        // 通过router配置参数注入路由，
        // 从而整个应用都有路由功能
```

```
        router // 缩写，向相当于router:router(es6语法)
    });
</script>
```

2. 路由导航

- 标签形式

```
<!--1. 标签形式-->
<router-link to='/home'>home</router-link>
```

我们也可以通过name属性来跳转

```
<router-link :to='{name:"settings"}'></router-link>
```

如果路由**携带参数**的话，通过 `/product/:id` 来设置路由

```
{name:'product',path:'/product/:id',component:product}
```

携带参数时路由跳转形式:
`<router-link v-bind='{to:"/product/"+item.id}'>首页</router-link>`

- js形式

```
<!--2. js形式-->
<button @click='toLogin'>login</button>
```

```
// 这里的'home'匹配的是router配置里的path
this.$router.push('home');
// 通过匹配name进行跳转
this.$router.push({name:'home'});
```

跳转路由时传递参数

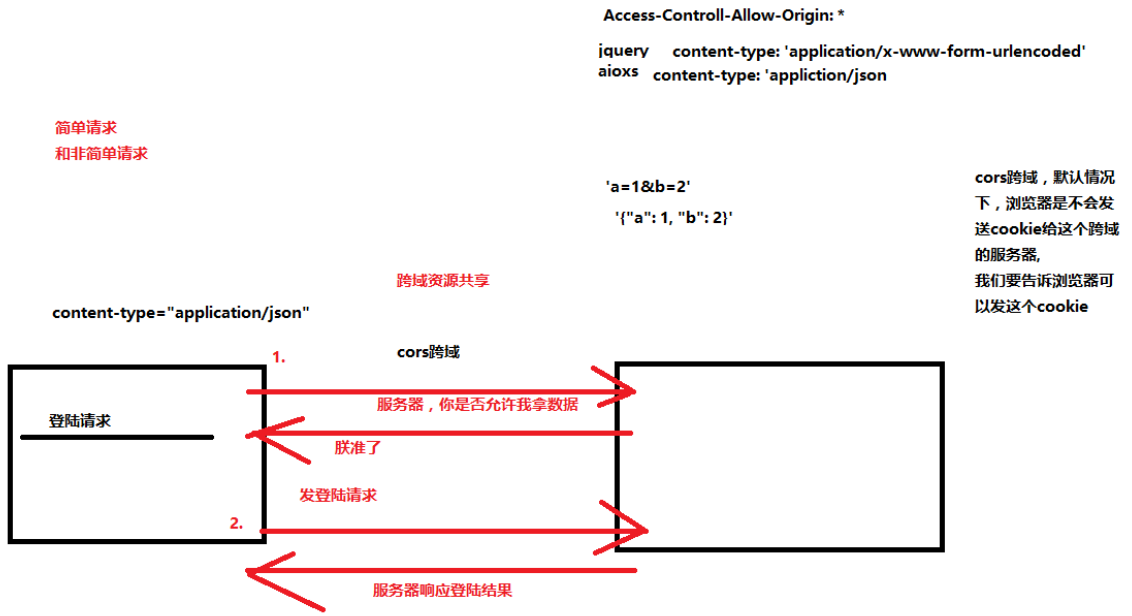
```
// 这里相当于/user/123
this.$router.push({name:'user',params:{userId:123}});
// 这里相当于/register?plan=private
this.$router.push({path:'register',query:{plan:'private'}});
```

- 接收参数

1. 在路由中通过 `/home/:id` 来配置路由: 参数被封装到了this.$route.params中 `this.$route.params.id`
2. 通过/home?id=123来传递参数: 参数被封装到了this.$route.query中 `this.$route.query.id`

3. cors(跨域资源共享)补充

cors跨域默认情况下浏览器不会发送cookie给服务器，我们通过设置 `withCredentials = true` 要求浏览器将cookie发送给服务器
否则服务器无法根据cookie将用户的信息存到session中



实例：统一让每一次的 `ajax` 请求的 `withCredentials` 为 `true` ，可以通过 `axios` 的拦截器

```
axios.interceptors.request.use(config=>{
// config:axios的配置信息
// 每次用axios发请求这个方法都会执行

// 要求浏览器将cookie一起发送给服务器
config.withCredentials = true;
    return config;
});
```

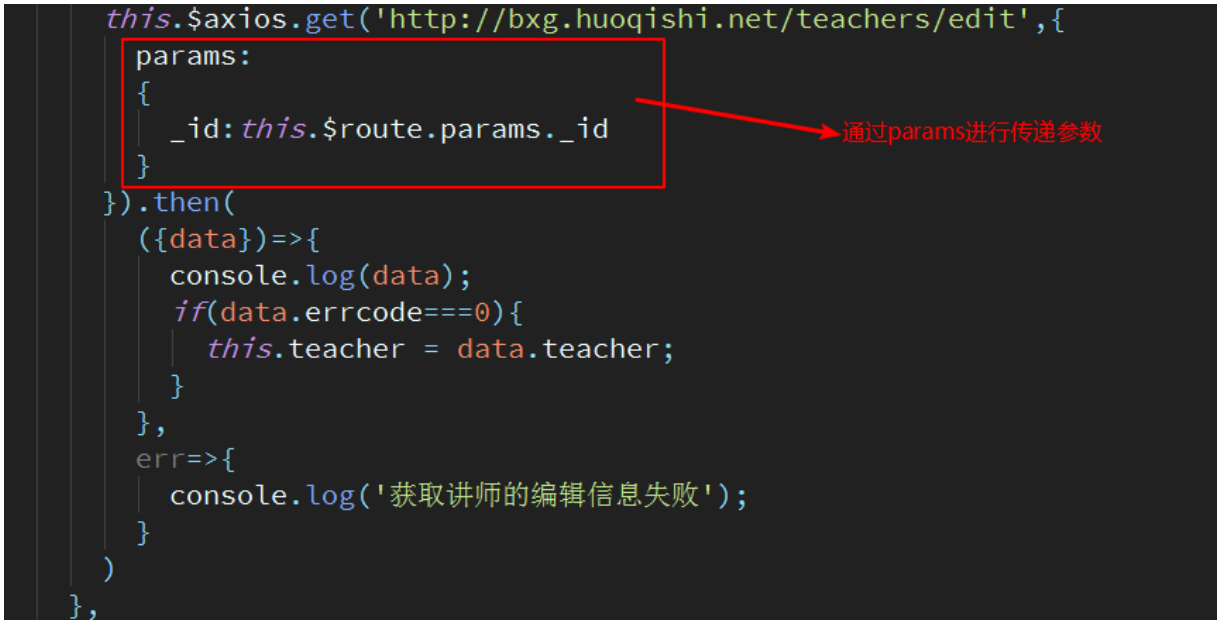
4. 为vue添加全局方法或属性

```
Vue.use(function (Vue) {
    // 给vue的每一个组件实例添加属性
    Vue.prototype.$axios = axios;
    // 在其它组件中直接通过this.$axios就可以调用axios的方法和属性
});
```

5. 使用axios请求数据

5.1 axios的传参方式

- get方式传参



- post方式传参

5.2 axios的拦截器

> `interceptors`:发起大量请求时，通过拦截器对请求和响应做统一处理

`axios` 发送 `ajax` 请求成功后返回的内容

```
const res = {
```

```
    // 服务器返回的数据
    data: {},
    // HTTP状态码
    status: 200,
    // 服务器返回的消息
    statusText: 'OK',
    // 返回头
    headers: {},
    // 返回我们的配置
    config: {}
}
```

统一的 `config` 配置

```
// 请求的超时时间是5秒
axios.defaults.timeout = 5000;
// 所有的请求都通过这个url走
// 可以更换线上环境和测试环境
axios.defaults.baseURL = "http://www.myshop.com/api";
// axios.defaults.baseURL = "http://192.168.1.129:8383";
// 设置post请求方式的请求头;
axios.defaults.headers.post['Content-Type'] = "application/x-www-form-urlencoded;charset=UTF-8";
```

- 在发起http请求的时候进行设置，每一个通过axios发起的请求都会执行这个方法

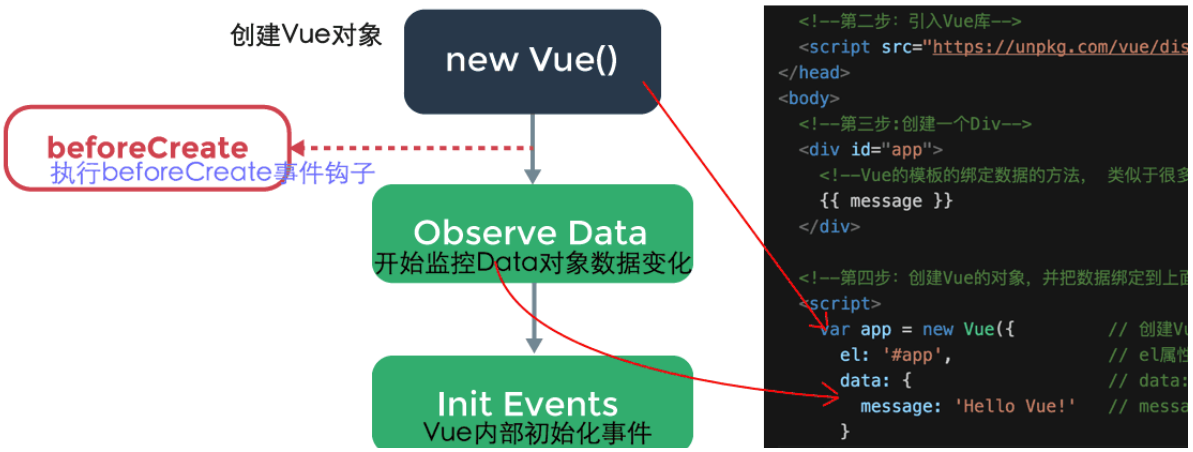
```
// 对请求做出统一处理
axios.interceptors.request.use(
    config => {
        // config:axios的配置信息
        // post传参序列化
        if (config.method === 'post') {
            // 将json格式的字符串进行key=value,key1=value2...的格式（序列化）
            config.data = qs.stringify(config.data);
            // console.log(config.data);
            // name=%E5%B0%8F%E6%98%8E&age=18
        }
        // 每次用axios发请求这个方法都会执行
        // 要求浏览器将cookie一起发送给服务器
        config.withCredentials = true;
        // 在请求之前开启进度条
        NProgress.done();
```

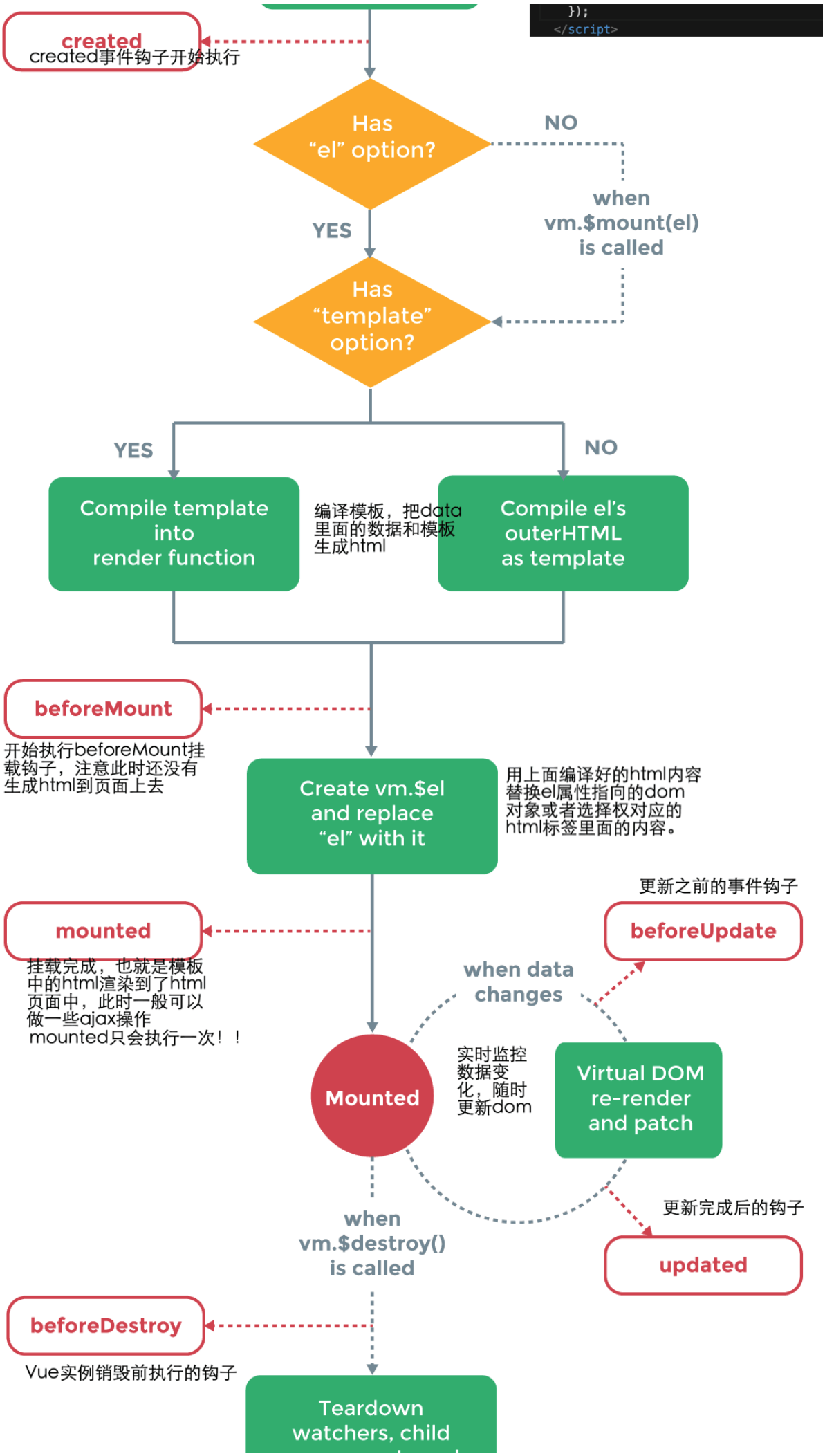
```
        // 可以在发请求之前对请求进行配置
        return config;
    },
    err => {
        alert('错误的传参');
        return Promise.reject(err);
    }
)
```

- 在收到服务器的响应的时候执行的内容

```
// 对响应做出统一处理
axios.interceptors.response.use(
    res => {
        // 这里的200是字符串（如果响应状态码不是为200）
        if (res.status != '200') {
            // 提示错误信息
            alert(res.data.msg);
            // 执行请求失败时的回调，支持promise API
            return Promise.reject(res);
        }
        // 响应成功后，加载完成进度条
        NProgress.done();
        return res;
    },
    err => {
        alert('网络异常');
        return Promise.reject(err);
    }
)
```

生命周期




});
</script>
created
created事件钩子开始执行
Has
“el” option?
NO
when
vm.$mount(el)
is called
YES
Has
“template”
option?
YES
NO
Compile template
into
render function
编译模板，把data
里面的数据和模板
生成html
Compile el’s
outerHTML
as template
beforeMount
开始执行beforeMount挂
载钩子，注意此时还没有
生成html到页面上去
Create vm.$el
and replace
“el” with it
用上面编译好的html内容
替换el属性指向的dom
对象或者选择权对应的
html标签里面的内容。
更新之前的事件钩子
mounted
beforeUpdate
挂载完成，也就是模板
中的html渲染到了html
页面中，此时一般可以
做一些ajax操作
mounted只会执行一次！！
when data
changes
Mounted
实时监控
数据变
化，随时
更新dom
Virtual DOM
re-render
and patch
更新完成后的钩子
when
vm.$destroy()
is called
updated
beforeDestroy
Vue实例销毁前执行的钩子
Teardown
watchers, child