# 之前的模式

1. 模式：客⼾端 -> http 请求 -> 服务端 -> 根据 url 路径的不同，返回不同的 html/数据
2. 优点：SEO效果好，首屏加载耗时短。
3. 缺点：服务器压力大，维护麻烦，协作流程不清晰--前后端没有完全分离

# spa

1. 不仅请求数据不刷新界面，而且切换路由也不刷新界面。而支持单页面应用这种特性的，就是前端路由。
2. 前端路由的需求：根据不同的URL展示不同的内容。不刷新界面。

# hash的原理及实现

1. hash的出现满足了前端的路由的这种需求
2. 特性：a、url中的hash值只是浏览器/客户端的一种状态，向服务器发送请求的时候，hash值不会携带。b、hash值的更改，不会导致页面的刷新。

c、hash值的更改，会在浏览器的访问历史中增加记录，所以我们可以通过浏览器的回退，前进按钮控制hash的切换。d、hash值的更改，会触发hashchange事件。

3）我们有两种方式控制hash值的变化：a、通过a标签设置href属性，当用户点击a标签的时候，url中的hash就会变为href的属性值

b、通过js：location.href= “#hash-change”

# 4、history

history虽然能解决问题，但是带有#不太美观。

window.history.back()

window.history.forward()

window.history.go(number)

window.history.pushState() => A B => A B C

window.history.replaceState() => A B=> A C

1. 可以使用popstate事件来监听url的变化。
2. pushState和replaceState并不会触发popstate事件。
3. 哪些情况会触发popstate呢？
4. 用户点击浏览器前进和回退按钮
5. history, back,forward go

部署history路由的时候，要记住，要使你的路由中的所有Path都访问到index.html

# 5、vue-router

vue router及异步组件

1）vue-router从列表浏览了一段时间，点击进了详情页。点击返回的时候，希望回到列表的时候还是停留在原来的位置，怎么做？

a、router-alive

b、// 1. router-link 的跳转, scrollBehavior没有记住滚动条的位置

// 2. 通过手动点击浏览器的返回或者前进按钮, 可以记住滚动条位置

// 新建一个router实例

const router = new VueRouter({

mode: "history", // hash，选择模式，history/hash

base: process.env.BASE\_URL,

routes, // 传入配置文件

scrollBehavior: (to, from, savedPostion) => {

console.log(savedPostion);

return savedPostion;

}

});

c、localStorage||sessionStorage scrollTo + scrollTop

d、注意：通过router-link的跳转，scrollBehavior没有记住列表的位置

通过主动点击浏览器的返回或者前进按钮的时候，scrollBehavior才会记住列表的位置

2）router-view是什么

路由出口---动态渲染模块

3）router同步加载和异步加载的区别

\*\* 同步：页面初始化的时候所有的模块代码都加载完

\*\* 异步：页面切换到对应的路由才去加载对应模块的代码

```js

import component1 from "../path"

const helloComponent = [

{

path: "/hello",

component: component1

}

]

```

```js

const helloComponent = [

{

path: "/hello",

component: () => {

import (/\*webpackChunkName:"about"\*/, "../hello")

}

}

]

```

4）导航守卫

```js

const helloComponent = [

{

path: "/hello",

component: () => {

import (/\*webpackChunkName:"about"\*/, "../path")

},

beforeEnter(to, from, next) { //路由钩子 执行顺序3

console.log(`${to}`)

next() //next函数必须执行

}

}

]

```

```js

<template>

</template>

<script>

export default{

beforeRouteEnter(to, from, next) { //执行顺序4

next()

}

beforeRouteUpdate(to, from, next) { //执行顺序 二级路由变化时执行

next()

}

beforeRouteLeave(to, from, next) { //离开当前界面时执行，顺序1

next()

}

}

</script>

```

```js

const router = new VueRouter({

routes,

})

router.beforeEach((to, from, next) => { //执行顺序2

console.log()

})

router.afterEach((to) => { //执行顺序 最后

console.log()

})

## 导航守卫的执行顺序

1. [组件] 前一个组件的beforeRouteLeave

2. [全局] router.beforeEach

3. [路由参数变化] beforeRouteUpdate

4. [配置文件里] beforeEnter

5. [组件] beforeRouteEnter

6. [全局] afterEach

``` 5）动态路由

/ 路由匹配的顺序是由上而下

const routes = [

{

path: "/", // 路径

name: "Home", // 名称

component: Home // 路径对应的组件/页面

},

{

// /dynamic/1111 :id

// /dynamic

// /dynamci/1111/lubai :id/:name

path: "/dynamic",

name: "Dynamic",

component: Dynamic,

children: [

{

path: "",

component: Dynamic

},

{

path: ":id",

component: Dynamic

},

{

path: ":id/:name",

component: Dynamic

}

]

},

{

path: "\*", // 这个路由可以匹配所有的路径 /list

name: "ErrorPage",

component: () =>

import(/\* webpackChunkName: "errorPage" \*/ "../views/ErrorPage.vue")

}

];

// 新建一个router实例

const router = new VueRouter({

mode: "history", // hash，选择模式，history/hash

base: process.env.BASE\_URL,

routes // 传入配置文件

});

6)路由监听

在App组件中

watch: {

“$router”: (from, to) =>{

}

}