Vue-router

课程目标：

P6：

P6+ ~P7：

课程大纲：

课程内容：

1. 什么是 Router，以及 Router 发展的历史。
2. 路由的分类

   hash 路由

   history 路由
3. Router 异步组件
4. 路由守卫

   路由守卫的触发流程
5. 路由的实现手写

   5.1 history 定义：

      路由系统最基本的，要包含当前的路径，以及在此路径下的状态

      除了路径以外，还要拿浏览器的状态

      跳转时，没有状态，所以我们需要传一些自己的状态

      实现一个 changeLocation 的方法

   5.2 push 和 replace 方法：

   5.3 完善一下方法的导出
6. 关于 前进和后退按钮的 Bug，我们需要怎么去处理。

   6.1 先增加一个 API，处理相关的问题

   6.2 创建 listener

   6.3 完善一下导出结果

题目总结

1. hash 路由 和 history 路由，有什么区别?
2. history.go / back 一定会刷新吗?
3. pushState 会触发 popState 事件吗?

4. 先来实现一个 hash 路由

课程目标：

P6：

- 针对 react / vue ，能够根据业务需求口喷 router 的关键配置，包括但不限于：路由的匹配规则、路由守卫、路由分层等。
- 能够描述清楚 history 的主要模式，知道 history 和 router 的边界；

P6+ ~P7:

- 在没有路由的情况下，也可以根据业务需要，实现一个简单的路由；
- 读过 router 底层的源码，不要求每行都读，可以口喷关键代码即可；

课程大纲：

- 路由发展的背景
- 实现简单的路由
- 讲解 vue-router 的相关用法：
  - 异步组件加载
  - 完整导航解析
  - 滚动行为
  - 路由守卫
- 彩蛋：讲解 vue-router 1:1 源码

课程内容：

0-43分： 路由的基础和面试题；
43-68分：简单 router 的编写；
68分-90分：动态组件；-webpack code splitting
90分-100分：路由守卫-具体可以看源码和官方API；
最后，讲一讲 vue-router 的一比一的源码。

1. 什么是 Router，以及 Router 发展的历史。

- 路由的概念，是伴随 SPA 出现的。在此之前，页面的跳转是通过服务器端进行控制的；

    - 传统的页面的跳转，是通过前端向后台发送请求
    - 后台通过模板引擎的渲染，将一个新的 html 界面
    - 比如页面跳转时：
        - from 表单的提交；
        - a 标签的默认属性；
        - js 调用 location.href，给其赋值；
        - H5: history 的 go / forward / back –– // history.push / replace ?
- 在 SPA（即只有一个 html ） 的出现后，前端可以**自由控制组件的渲染**，来**模拟**页面的跳转。
    - 页面是怎么发生跳转，向服务端请求的呢？–– 浏览器劫持。
    - 在讲这部分内容前，我们先来说一下，hash 路由和 history 路由的区别
    - SPA的方法，需要拦截请求；
        - hash 路由，当我的hash
        - history 的 go / forward / back 的时候，我的浏览器的地址，是发生了改变的，

总结：

- 后端路由是根据 `url` 访问相关的 `controller` 进行数据资源和模板引擎的拼接，返回前端；
- 前端路由是通过 `js` 根据 `url` 返回对应的组件加载。
- 所以，前端的路由包含两个部分：
    - `url` 的处理
    - 组件加载

2. 路由的分类

- history 路由
- hash 路由
- memory 路由 *

hash 路由

`window.location.hash = "xxx"`

history 路由

`history./\(go|back|repalce|push|forward)/`

3. Router 异步组件

其实，动态路由 包括 `React.lazy` 、 `import()` 就是一种对代码进行动态拆分的技术，我们一般叫做 `code splitting` 。
在需要的时候，才进行加载；

4. 路由守卫

路由守卫的触发流程

1. 【组件】– 前一个组件 `beforeRouteLeave`
2. 【全局】– `router.beforeEach`
3. 【组件】–如果是路由的参数变化，触发 `beforeRouteUpdate`；
4. 【配置文件】里，下一个 `beforeEnter`
5. 【组件】内部声明的 `beforeRouteEnter`
6. 【全局】调用 `beforeResolve`
7. 【全局】的 `router.afterEach`

5. 路由的实现手写

在新的版本中，由于浏览器当前对H5的兼容性的保证，所以，我们就用 history ，来实现两种路由。
我们先实现 H5，然后让他去兼容 hash 就可以了。
这也是 最新的 vue/router 的典型写法。

5.1 history 定义：

路由系统最基本的，要包含当前的路径，以及在次路径下的状态，
同时要实现路由监听，如果路径变化，需要通知用户。

路由系统最基本的，要包含当前的路径，以及在此路径下的状态

JavaScript | 复制代码

```
const history = createWebHistory()
// 创建一个历史导航
function createWebHistory() {
  // 创建一个对象，包含路径、状态、以及push/replace 切换的方法
  const historyNavigation = useHistoryStateNavigation();
}
```

```JavaScript
function useHistoryStateNavigation() {
  // const currentLocation = '/' // 更改的时候，尽量是一个引用类型。
  const currentLocation = {
    // 如何获取路径呢？用 window.location 的结果去拼接，我们再封装一个方法
    value: createCurrentLocation();
  }
}

function createCurrentLocation() {
  const {pathname, search, hash} = window.location;
  return pathname + search + hash;
}
```

除了路径以外，还要拿浏览器的状态

```JavaScript
function useHistoryStateNavigation() {
  // const currentLocation = '/' // 更改的时候，尽量是一个引用类型。
  const currentLocation = {
    // 如何获取路径呢？用 window.location 的结果去拼接，我们再封装一个方法
    value: createCurrentLocation()
  }
  // 除了路径以外，还要拿浏览器的状态
  const historyState = {
    value: window.history.state
  }

  console.log(currentLocation, historyState);
}
// 测试一把
```

跳转时，没有状态，所以我们需要传一些自己的状态

比如第一次刷新界面时，没有任何状态，我就自己去维护一个状态。

- forward, back, current, scroll,

JavaScript | 复制代码

```javascript
function buildState(back, current, forward, replace = false,
computedScroll = false) {
  return {
    back, current, forward, replace, scroll: computedScroll ?
    {left: window.pageXOffset, top: window.pageYOffset}: null,
    // 还要记录一下，当前是第几层
    position: window.history.length -1 // 默认从2开始，所以我们 -1
  }
}

function useHistoryStateNavigation() {
  // const currentLocation = '/' // 更改的时候，尽量是一个引用类型。
  const currentLocation = {
    // 如何获取路径呢？用 window.location 的结果去拼接，我们再封装一个方法
    value: createCurrentLocation();
  }
  // 除了路径以外，还要拿浏览器的状态
  const historyState = {
    value: window.history.state
  }

  ////////////////
  if(!historyState.value) {
    // 但是你这样改，是不会更新你的 historyState 的，没有同步到路由系统中。
    buildState(null, currentLocation.value, null, true);
    // 打印一下。
  }
  console.log(currentLocation, historyState);
}
// 测试一把
```

但是你这样改，是不会更新你的 historyState 的，没有同步到路由系统中。

所以，我们：

实现一个 changeLocation 的方法

传入 去哪儿/状态和 `repalce` ？给我

JavaScript | 复制代码

```javascript
function useHistoryStateNavigation() {
  // const currentLocation = '/' // 更改的时候，尽量是一个引用类型。
  const currentLocation = {
    // 如何获取路径呢？用 window.location 的结果去拼接，我们再封装一个方法
    value: createCurrentLocation();
  }
  // 除了路径以外，还要拿浏览器的状态
  const historyState = {
    value: window.history.state
  }

  ////////////////
  if(!historyState.value) {
    // 但是你这样改，是不会更新你的 historyState 的，没有同步到路由系统中。
    changeLocation(
      currentLocation.value,
      buildState(null, currentLocation.value, null, true),
      true
    )
    // 打印一下。
  }

  console.log(currentLocation, historyState);

  function changeLocation(to, state, replace) {
    window.history[replace?'replace':'pushState'](state, null, to);
    historyState.value = state;
  }
}

```

到这个时候，我的核心，就变成了
`currentLocation` 和 `historyState`；
相当于是路由中的 `location` 和 `history`

5.2 push 和 replace 方法：

在哪里写？ `useHistoryNavigation`

JavaScript 复制代码

```javascript
function push(to, data) {
  // 去哪，带的新的状态是谁?

  // 跳转前：从哪儿，去哪儿
  // why ? 为了做路由守卫。
  const currentState = Object.assign(
    {},
    historyState.value,
    // 只需要改去哪儿，和当前的滚动条的位置。
    {forward: to,  scroll: {left: window.pageXOffset, top: window.pageYOffset}}
  )
  // 本质是没有跳转的，只是，更新了状态，后续在 Vue 中，可以监控到详细的状态变化。
  // 所以这里是 replace 模式
  changeLocation(currentState.current, currentState, true);

  // 跳转后：从这儿到哪儿
  const state = Object.assign(
    {},
    buildState(currentLocation.value, to, null),
    {position: currentState.position+1},
    data,
  )

  // 这里要真正的跳转，所以是 push 模式
  changeLocation(to, state, false);
  currentLocation.value = to;

}
function replace(to, data){
  // 创建一个状态，并进行合并
  // 只要替换掉 current 即可。
  const state = Object.assign(
    {},
    buildState(historyState.value.back, to, historyState.value.forward, true),
    data
  )

  changeLocation(to, state, true);
  // 这一步是维护我自己状态的，替换后需要将路径变为现在的路径，就不再用原生的了
  // 用我自己的，我的功能更强大一些。
  currentLocation.value = to;
}
```

5.3 完善一下方法的导出

JavaScript 复制代码

```
function useHistoryStateNavigation() {

  // .....

  return {
    location: currentLocation,
    state: historyState,
    push,
    replace
  }
}

function createWebHistory() {
  return {
    ...historyNavigation
  }
}
```

6. 关于 前进和后退按钮的 Bug，我们需要怎么去处理。

6.1 先增加一个 API，处理相关的问题

JavaScript | 复制代码

```javascript
function useHistoryListeners(historyState, currentLocation) {
  const popStateHandler = ({state}) => {
    const to = createCurrentLocation(); // 去哪
    const from = currentLocation.value; // 从哪儿来
    const prevState = historyState.value;
    // 开始修改啦
    currentLocation.value = to;
    historyState.value = state;

    let isBack = state.position - prevState.position < 0;

    // 用户扩展的地方，就在这里，也就是连接 vue 组件和 history 之间的核心。

    // !!!!!!!!!!!!!!!   !!!!!!!!!!!!!!!!

  }

  window.addEventListener('popstate', popStateHandler)
}

// 创建一个历史导航
function createWebHistory() {
  // 创建一个对象，包含路径、状态、以及push/replace 切换的方法
  const historyNavigation = useHistoryStateNavigation();

  // 构建一个监听函数，监听浏览器的前进和后退
  const {location, state} = historyNavigation;
  const historyListeners = useHistoryListeners(state, location);
  return {
    ...historyNavigation
  }
}
```

6.2 创建 listener

JavaScript | 复制代码

```javascript
function useHistoryListeners(historyState, currentLocation) {

  let listeners = [];

  const popStateHandler = ({state}) => {
    const to = createCurrentLocation(); // 去哪
    const from = currentLocation.value; // 从哪儿来
    const prevState = historyState.value;
    // 开始修改啦
    currentLocation.value = to;
    historyState.value = state; // state 可能为空

    let isBack = state.position - prevState.position < 0;

    // 用户扩展的地方，就在这里，也就是连接 vue 组件和 history 之间的核心。
    listeners.forEach(listener => {
      listener(to, from, {isBack})
    })

  }

  window.addEventListener('popstate', popStateHandler);

  function listen(cb) {
    listeners.push(cb);
  }

  return {
    listen
  }
}

// 创建一个历史导航
function createWebHistory() {
  // 创建一个对象，包含路径、状态、以及push/replace 切换的方法
  const historyNavigation = useHistoryStateNavigation();

  // 构建一个监听函数，监听浏览器的前进和后退
  const {location, state} = historyNavigation;
  const historyListeners = useHistoryListeners(state, location);
  return {
    ...historyNavigation,
    listen: historyListeners.listen
  }
}
```

```
46
47  const webHistory = createWebHistory();
48
49  webHistory.listen((to, from, {isBack}) => {
50    console.log(to, from, {isBack})
51  })
```

6.3 完善一下导出结果

JavaScript | 复制代码

```
1  function createWebHistory() {
2    // 创建一个对象，包含路径、状态、以及push/replace 切换的方法
3    const historyNavigation = useHistoryStateNavigation();
4
5    // 构建一个监听函数，监听浏览器的前进和后退
6    const {location, state} = historyNavigation;
7    const historyListeners = useHistoryListeners(state, location);
8
9    const routerHistory = Object.assign(
10     {},
11     historyNavigation,
12     historyListeners
13   )
14
15   Object.defineProperty(routerHistory, 'location', {
16     get: () => historyNavigation.location.value
17   })
18
19   Object.defineProperty(routerHistory, 'state', {
20     get: () => historyNavigation.state.value
21   })
22
23   return routerHistory
24 }
25
26 const webHistory = createWebHistory();
27
```

题目总结

1. hash 路由 和 history 路由，有什么区别?

- hash 路由 一般会携带 一个 # 号，不够美观； history 路由不存在这个问题；
- 默认 hash 路由是不会像浏览器发出请求的，主要是一般用于锚点；history 中 go / back / forward 以及浏览器的前进、后退按钮**一般**都会像服务端发起请求；-- **history 的所有 url 内容，服务端都可以获取到**
- 基于此，hash 模式，是不支持SSR的，但是 history 模式可以做 SSR
- history 在部署的时候，如 nginx， 需要只渲染首页，让首页根据路径重新跳转。
- 要注意：如何部署

Nginx 复制代码

```nginx
# 单个的服务器部署
location / {
  try_files uri $uri /xxx/main/index.html
}

# 存在代理的情况
location / {
  rewrite ^ /file/index.html break; # 这里代表的是xxx.cdn 的资源路径
  proxy_pass https://www.xxx.cdn.com;
}
```

2. history.go / back 一定会刷新吗?

- 要根据指定页面和当前界面的构建关系，动态决定；

3. pushState 会触发 popState 事件吗?

popState 是监听其他的操作。

- pushState/replaceState 都不会触发 popState 事件，需要触发页面的重新渲染。
- popState 什么时候触发?
  - 点击浏览器的前进、后退按钮
  - back / forward / go

4. 先来实现一个 hash 路由

1. 一起写一个 vue-router 的简单实现
2. 一起做一些 vue-router 的实战和关键点、面试题

3. 一起看一下 vue-router 的源码