1. 前端路由 router 原理及表现

- 核心都是改变url，但不刷新页面，不向服务器发送请求

1.1 hash路由

- url 的 hash 是以 # 开头，当 hash 改变时，页面不会因此刷新，浏览器也不会向服务器发送请求。
- 特点：兼容性好、丑陋、对于后端路由来说不区分#号后面的内容
  - http://a.com/web#order
  - http://a.com/web#goods
  - 以上两个路由对于后端来说没有区别，都是/web路径下的，所以不需要后端特殊支持
- 更改hash及hashchange事件

```
location.hash = '#/news'
location.replace('#/detial') // 替换当前记录
// https://www.baidu.com  -> https://www.baidu.com/#news
// 同时在浏览器生成一条记录，点击回退按钮会回到原url

// 监听hash的变化，显示不同的内容
window.addEventListener('hashchange', function () {
  // 隐藏掉之前路由的内容
  // 显示当前路由的内容
  // document.querySelector('#root').innerHTML = '当前hash内容'
  console.log('render');
});
```

- hash 路由例子

```
  <div>
    <div>
      <h1>hash 路由</h1>
      <a href="#/list">列表页</a>
      <a href="#/detail">详情页</a>
      <a href="#/other">404</a>
    </div>
    <div id="app" style="border: 1px solid black; min-height: 200px;"></div>
  </div>


<script>

  // 定义路由映射表
  var routerObj = {
    '#/list': '<div>列表页</div>',
    '#/detail': '<div>详情页</div>'
  }

  window.addEventListener('hashchange', function() {
    // 监听hash路由变化
    // 拿到映射对应的组件进行渲染
    document.getElementById('app').innerHTML = routerObj[location.hash] || '404页面'
  })

</script>
```

1.2 history路由

- HTML5 规范中提供了 history.pushState 和 history.replaceState 来进行路由控制。通过这两个方法，

可以实现改变 url 且不向服务器发送请求

- history api demo

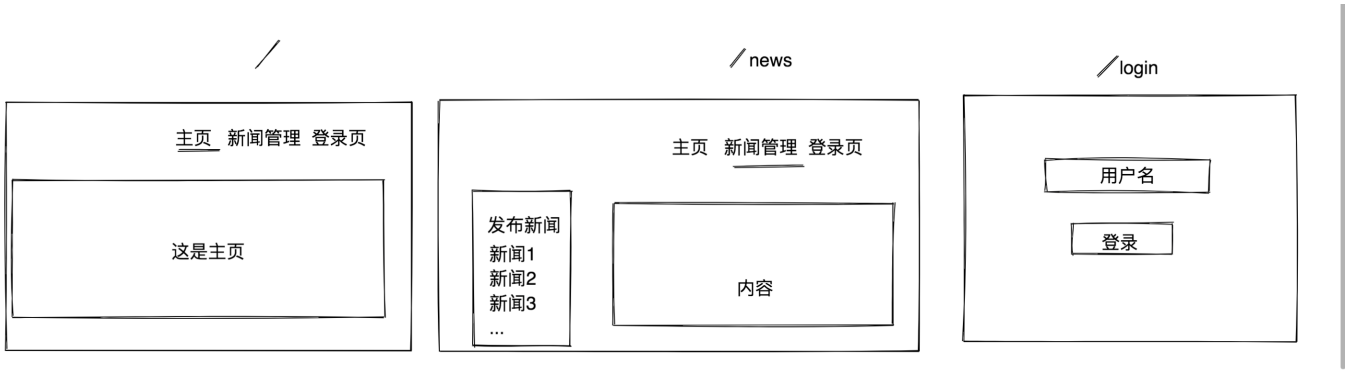
```
history.pushState({}, '', '/news')
// https://www.baidu.com -> https://www.baidu.com/news
// 同时在浏览器生成一条记录，点击回退按钮会回到原url(repalceState会覆盖掉当前的记录)
```

- 需要服务端配合，避免刷新404
  - http://a.com/web/order
  - http://a.com/web/goods
  - 对于后端来说可能是两个页面，要做一个通配符识别，将/web/*后面的统一返回某个html中
- history路由没有hash路由类似的`hashchange`事件。
- 改变当前url有两种方式
  1. 点击后退/前进 -> `popstate`事件
  2. history.pushState history.replaceState -> 触发相应的函数后，在后面手动添加回调
- history 路由 demo

```
<div id="history-box">
  <h1>history 路由</h1>
  <a href="/web/list">列表页</a>
  <a href="/web/detail">详情页</a>
  <a href="/web/other">404</a>
</div>

<script>
  // history 路由demo
  var routerHistoryObj = {
    '/web/list': '<div>history 列表页</div>',
    '/web/detail': '<div>history 详情页</div>'
  }

  // 为每个链接添加点击事件
  var length = document.querySelectorAll('#history-box a[href]').length
  for(var i = 0; i < length; i++) {
    document.querySelectorAll('#history-box a[href]')[i].addEventListener('click', function(event) {
      event.preventDefault();
      window.history.pushState({}, null, event.currentTarget.getAttribute('href') );
      handleHref();
    })
  }


  // 监听前进/后退 引起的posstate事件
  window.addEventListener('popstate', handleHref);

  // 根据新的路由，显示新的组件
  function handleHref () {
    document.getElementById('app').innerHTML = routerHistoryObj[location.pathname] || '404页面'
  }

</script>


// nodejs路由处理 /web -> /web*
app.get('/web*', (req, res) => {
  res.sendFile(__dirname + '/index.html');
})
```

2. vue.js router 使用详解

2.1 介绍DEMO



2.2 引入vue-router并注册插件

```
Vue.use(VueRouter)
// 引入了两个组件 router-link和router-view，及全局混入了$route $router
```

2.2.1 router-link 用法

to 字符串 | Location对象

- 字符串,手动拼接的
- {name: '', query: {}, params: {}}
- {path: '', query: {}, params: {}}

- tag 默认为a
- repalce
- 与手写a链接的区别，router-link抹平了两种模式下href的书写方式，会得到正确的href值；history模式下调用pushState并阻止默认行为。

2.2.2 router-view

- 确定路由组件显示的位置
- 可以嵌套
- 命名router-view

```
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>


const router = new VueRouter({
  routes: [
    {
      path: '/',
      components: {
        default: Foo,
        a: Bar,
        b: Baz
      }
    }
  ]
})
```

2.2.3 this.$route

- params
- query
- matched // 匹配的路由记录
- path

2.2.4 this.$router

- push(location)
- replace(location)
- go(n)
- back()
- forward()
- resolve()
    - const {href} = this.$router.resolve(location) // 得到完整的url，可以window.open打开

2.3 简单的routes和component demo

```
// 0. 注册插件 Vue.use(VueRouter)
// 1. 定义 (路由) 组件。
// 可以从其他文件 import 进来
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2. 定义路由
// 每个路由应该映射一个组件。
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3. 创建 router 实例，然后传 `routes` 配置

const router = new VueRouter({
  routes // (缩写) 相当于 routes: routes
})

// 4. 创建和挂载根实例。
// 记得要通过 router 配置参数注入路由，
// 从而让整个应用都有路由功能
const app = new Vue({
  router
}).$mount('#app')
```

2.4 命名路由

- 可以直接通过名字跳转，后续如果更改了path，则不影响name的跳转
- 设置了默认的子路由，则子路由的name会被警告，通过name跳转子路由则不会显示默认的子路由设置了默认的子路

2.5 子路由

- 默认子路由：path: ''
- 子路由中的path是否以'/'开头的区别，加'/'是绝对路径，不加是相对

```
children: [
    {
        path: '',
        component: NewsAdd,
        name: 'newsDefault'
    },
    {
        path: 'add',
        component: NewsAdd,
        name: 'newsAdd'
    },
    {
        path: 'detail/:id',
        component: NewsDetail,
        name: 'newsDetail',
    }
]
```

2.6 动态匹配路由

params: `/user/:username`

2.6.1 响应路由参数变化

- watch

```
watch: {
  '$route.params.id'() {
      this.getNews()
  }
},
```

- beforeRouteUpdate v2.2

```
beforeRouteUpdate(to, from, next) {
  this.getNews(to.params.id)
  next()
},
```

2.7 404路由

```
// 含有通配符的路由应该放在最后
{
    path: '*',
    component: NotFound,
},
```

2.8 导航守卫

2.8.1 全局守卫

- 前置守卫：beforeEach(to, from, next)
    - 必须调用next()才可继续
    - next('/') next({path: '/'}) 当前的导航被中断，然后进行一个新的导航。比如访问需要登录的页面，如果没有登录的话， 就跳转到登录页

- 解析守卫: beforeResolve(to, from, next)
    - 2.5.0新增
    - 组件内守卫和异步路由组件被解析之后，导航被确认之前被调用
- 后置守卫: afterEach(to, from)
    - 无next参数，不会改变导航，因为导航已被确认

2.8.2 路由独享守卫

- beforeEnter

```
const router = new VueRouter({
  routes: [
      {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
          // ...
      }
      }
  ]
})
```

2.8.3 组件守卫

- beforeRouteEnter(to, from, next)
    - 在渲染该组件的对应路由被 confirm 前调用
    - 不能访问this，组件实例还未被创建
    - 可以给next传递一个回调访问this，也是唯一一个支持给next传递回调的守卫

```
beforeRouteEnter (to, from, next) {
  next(vm => {
      // 通过 `vm` 访问组件实例
  })
}
```

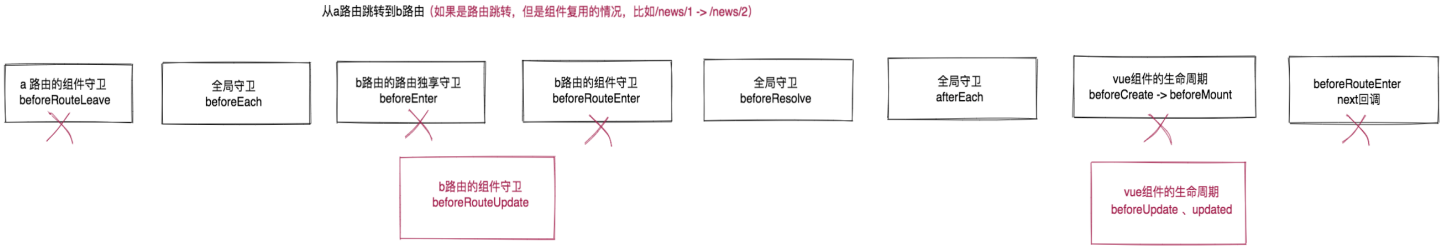
- beforeRouteUpdate(to, from, next)
    - 在当前路由改变，但是该组件被复用时调用
    - 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

```
beforeRouteUpdate (to, from, next) {
  // just use `this`
  this.name = to.params.name
  next()
}
```

- beforeRouteLeave(to, from, next)
    - 导航离开该组件的对应路由时调用
    - 这个离开守卫通常用来禁止用户在还未保存修改前突然离开。该导航可以通过 next(false) 来取消。

```
beforeRouteLeave (to, from, next) {
  const answer = window.confirm('您确定离开吗？还有未保存的更改')
  if (answer) {
      next()
  } else {
      next(false)
  }
}
```

2.8.4 完整的导航解析流程



2:9 路由元信息meta

- 比如在路由鉴权中使用

  $route.matched来检查路由中的字段，因为路由是可以嵌套的，一个路由匹配成功后，可能会匹配多个路由记录

```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    // this route requires auth, check if logged in
    // if not, redirect to login page.
    if (!auth.loggedIn()) {
      next({
        path: '/login',
        query: { redirect: to.fullPath }
      })
    } else {
      next()
    }
  } else {
    next() // 确保一定要调用 next()
  }
})
```

3. 路由懒加载及异步组件

- 优点：在这个组件需要被渲染的时候才会触发该工厂函数
- 路由懒加载，异步组件需要一个函数

```
const Foo = () => import('./Foo.vue')
// /* webpackChunkName: "group-foo" */
const router = new VueRouter({
  routes: [
      { path: '/foo', component: Foo }
  ]
})
```

- 异步组件

```
new Vue({
  // ...
  components: {
      'my-component': () => import('./my-async-component')
  }
})
```

- vue-cli3默认支持。在webpack中需要使用syntax-dynamic-import 插件，才能使babel支持

- prefetch: vue-cli3 对动态import()生成的资源 自动添加prefetch，当前页面可能会用到的资源，在浏览器空闲时加载
- preload: vue-cli3 应用会为所有初始化渲染需要的文件自动生成preload，用来指定页面加载后很快会被用到的资源

4. 常见面试题

- 如何重定向页面

```
const router = new VueRouter({
  routes: [
      { path: '/a', redirect: '/b' }
  ]
})
```

- 路由有几种模式？说说它们的区别？
- 讲一下完整的导航守卫流程？
- 路由导航守卫和Vue实例生命周期钩子函数的执行顺序？
- 路由组件和路由为什么解耦，怎么解耦？
  - 解耦前

```
const Home = {
  template: '<div>User {{ $route.params.id }}</div>'
}
const router = new VueRouter({
    routes: [
        { path: '/home/:id', component: Home }
    ]
})
```

使用props解耦后，props为true，route.params将会被设置为组件属性。

```
const Home = {
  props: ['id'],
  template: '<div>User {{ id }}</div>'
}
const router = new VueRouter({
    routes: [
        { path: '/home/:id', component: Home, props: true},
    ]
})
```

- 直接使用a链接与使用router-link的区别
- Vue路由怎么跳转打开新窗口？

```
const obj = {
    path: xxx,//路由地址
    query: {
      mid: data.id//可以带参数
    }
};
const {href} = this.$router.resolve(obj);
window.open(href, '_blank');
```

5. 作业

- 将vue-router选项扁平化处理(将一颗嵌套的树扁平化处理，BFS, DFS)

```
[
  {
    path: '/',
    component: Index,
  },
  {
    path: '/news',
    component: News,
    children: [
      {
        path: 'detial',
        component: Detail,
        children: [....]
      }
    ]
  },
}
```

```
{
  '/': {path, component, ...},
  '/news': ...,
  '/news/detail': ...,
}
```