

vue-router를 이용한 라우팅



■ 여러개 화면을 가진 앱을 개발하려면...

- 동적 컴포넌트를 이용하는 방법
 - 어느 정도 해결이 가능하지만...
 - 특정 화면으로 바로 이동할 수 없고, 반드시 초기화면을 거쳐야만 함.
- 라우팅 기능 적용

■ SPA(Single Page Application)

- 최근의 웹애플리케이션 구조
- 여러개 화면을 하나의 페이지 안에서 제공하면서도 화면마다 페이지를 작성하지 않음.
- URI를 고유 식별자로 이용해 특정 화면을 보여줌
- vue에서는 vue-router 구성 요소를 사용함.

vue-router를 이용한 라우팅



■ vue-router의 특징

- 중첩된 경로, 뷰를 맵핑
- 컴포넌트 기반의 라우팅 구현
- vue.js의 전환 효과(Transition) 적용 가능
- 히스토리 모드와 해시 모드 둘 다 사용.
- 쿼리스트링, 파라미터, 와일드 카드를 사용하여 라우팅 구현 가능



■ routertest 프로젝트

```
vue init webpack-simple routertest  
cd routertest  
npm install  
npm install --save vue-router
```

■ 9장 4절을 참조해 다음 컴포넌트 추가

- Home.vue, About.vue, Contacts.vue
 - 볼드체로 표현한 부분만 컴포넌트마다 다르게 지정

```
<template>  
  <div>  
    <h1>Home</h1>  
  </div>  
</template>  
<script>  
export default {  
  name : "home"  
}  
</script>
```

vue-router 기초



■ 예제 12-02 : src/App.vue

- 기존 내용을 삭제하고 새롭게 작성

```
<template>
<div>
  <div class="header">
    <h1 class="headerText">(주)OpenSG</h1>
  <nav>
    <ul>
      <li><router-link to="/home">Home</router-link></li>
      <li><router-link to="/about">About</router-link></li>
      <li><router-link to="/contacts">Contacts</router-link></li>
    </ul>
  </nav>
</div>

<div class="container">
  <router-view></router-view>
</div>
</template>
```

```
<script>
import Home from './components/Home.vue';
import About from './components/About.vue';
import Contacts from './components/Contacts.vue';
import VueRouter from 'vue-router';

const router = new VueRouter({
  routes : [
    { path: '/', component: Home },
    { path: '/home', component: Home },
    { path: '/about', component: About },
    { path: '/contacts', component: Contacts }
  ]
})

export default {
  name : 'app',
  router
}

</script>
<style>
.....(생략)
</style>
```

vue-router 기초



■ 예제 12-03 : src/main.js

```
import Vue from 'vue'  
import App from './App.vue'  
import VueRouter from 'vue-router';  
  
Vue.use(VueRouter);  
  
new Vue({  
  el: '#app',  
  render: h => h(App)  
})
```

vue-router 기초



■ 실행 결과

- route 객체 정보
 - params, query, fullPath 정보 등을 활용할 수 있음.

The screenshot shows a browser window with the title "routerest" and URL "localhost:8080/#/contacts". The page content displays a header with "(주)OpenSG" and a navigation bar with "Home", "About", and "Contacts". The main area is titled "Contacts".

The browser's developer tools are open, specifically the Vue DevTools. The "Elements" tab is selected. In the component tree, under the <App> component, there are three <RouterLink> components and one <Contacts> component. The <Contacts> component is highlighted with a green background and has a status bar below it showing "\$vm0" and "router-view: /contacts".

In the bottom right panel, the "data" section of the component is expanded, showing the following object structure:

```
data
  ↓ $route: Object
    fullPath: "/contacts"
  ↓ meta: Object (empty)
  ↓ params: Object (empty)
    path: "/contacts"
  ↓ query: Object (empty)
```

동적 라우트



■ Dynamic Route

- 일정한 패턴의 URI 경로를 하나의 컴포넌트 연결하는 방법
- URI 경로의 일부에 실행에 필요한 파라미터 값이 포함된 경우에 유용함

■ 기존 프로젝트에 이어서 코드 작성

- 추가할 기능
 - contacts 경로로 요청했을 때 전체 연락처 명단을 나타내고,
 - 명단을 클릭하면 /contacts/{no} 경로로 요청하여 연락처 상세 정보를 나타냄



■ 예제 12-04 : src/ContactList.json(샘플데이터)

```
var contactlist = {  
    contacts : [  
        { no:1001, name:'김유신', tel:'010-1212-3331', address:'경주' },  
        { no:1002, name:'장보고', tel:'010-1212-3332', address:'청해진' },  
        { no:1003, name:'관창', tel:'010-1212-3333', address:'황산벌' },  
        { no:1004, name:'안중근', tel:'010-1212-3334', address:'해주' },  
        { no:1005, name:'강감찬', tel:'010-1212-3335', address:'귀주' },  
        { no:1006, name:'정봉주', tel:'010-1212-3336', address:'개성' },  
        { no:1007, name:'이순신', tel:'010-1212-3337', address:'통제영' },  
        { no:1008, name:'김시민', tel:'010-1212-3338', address:'진주' },  
        { no:1009, name:'정약용', tel:'010-1212-3339', address:'남양주' }  
    ]  
}  
  
export default contactlist;
```



■ 예제 12-05 : src/components/Contacts.vue

```
<template>
<div>
  <h1>연락처</h1>
  <div class="wrapper">
    <div class="box" v-for="c in contacts" :key="c.no">
      <router-link v-bind:to="/contacts/"+c.no>{{c.name}}</router-link>
    </div>
  </div>
</div>
</template>
<script>
import contactlist from '../ContactList';

export default {
  name : "contacts",
  data : function() {
    return {
      contacts : contactlist.contacts
    }
  }
}
</script>
<style>
.....(생략)
</style>
```

동적 라우트



■ 예제 12-06 : src/components/ContactByNo.vue

```
<template>
<div>
  <h1>연락처 상세</h1>
  <div >
    <table class="detail table table-bordered">
      <tbody>
        <tr class="active">
          <td>일련번호</td>
          <td>{{contact.no}}</td>
        </tr>
        <tr class="active">
          <td>이름</td>
          <td>{{contact.name}}</td>
        </tr>
        <tr class="active">
          <td>전화</td>
          <td>{{contact.tel}}</td>
        </tr>
        <tr class="active">
          <td>주소</td>
          <td>{{contact.address}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
</template>
```

```
<script>
import contactlist from '../ContactList';
export default {
  name : 'contactbyno',
  data : function() {
    return {
      no : 0,
      contacts : contactlist.contacts
    }
  },
  created : function() {
    this.no = this.$route.params.no;
  },
  computed : {
    contact : function() {
      var no = this.no;
      var arr = this.contacts.filter(function(item, index) {
        return item.no == no;
      });
      if (arr.length == 1)  return arr[0];
      else  return {};
    }
  }
}
</script>
<style>.....(생략)</style>
```

동적 라우트



■ 예제 12-07 : src/App.vue 변경

```
<template>
.....(변경사항 없음)
</template>
<script>
import Home from './components/Home.vue';
import About from './components/About.vue';
import Contacts from './components/Contacts.vue';
import ContactByNo from './components/ContactByNo.vue';
import VueRouter from 'vue-router';

const router = new VueRouter({
  routes : [
    { path: '/', component: Home },
    { path: '/home', component: Home },
    { path: '/about', component: About },
    { path: '/contacts', component: Contacts },
    { path: '/contacts/:no', component: ContactByNo }
  ]
})

export default {
  name : 'app',
  router
}
</script>
<style>.....(생략) </style>
```

동적 라우트



실행 결과

(주) OpenSG

Home About Contacts

연락처

김유신 정보고 권창 안중근
김경진 정몽주 이승신 김시민
정의용

Ready. Detected Vue 2.3.4.

<Root>

- <App>
 - <RouterLink>
 - <RouterLink>
 - <RouterLink>
 - <Contacts> == \$vm0 router-view: /contacts

data

- \$route: Object
 - fullPath: "/contacts"
 - meta: Object (empty)
 - params: Object (empty)
 - path: "/contacts"
 - query: Object (empty)
 - contacts: Array[9]



(주) OpenSG

Home About Contacts

연락처 상세

일련번호	1007
이름	이승신
전화	010-1212-3337
주소	동계영

Ready. Detected Vue 2.3.4.

<Root>

- <App>
 - <RouterLink>
 - <RouterLink>
 - <RouterLink>
 - <Contactbyno> == \$vm0 router-view: /contact

data

- \$route: Object
 - fullPath: "/contacts/1007"
 - meta: Object (empty)
 - params: Object
 - path: "/contacts/1007"
 - query: Object (empty)
 - contacts: Array[9]
 - no: "1007"

동적 라우트



■ \$route 객체를 이용해 획득할 수 있는 정보

- params
 - /contacts/:no → this.\$route.params.no
- query
 - /users?keyword=test → this.\$route.query.keyword
- fullPath : "/contacts/1002"



▪ Nested Route

- 컴포넌트들이 중첩되는 경우 라우팅도 중첩 가능해야 함.
- 한 컴포넌트 내부에 다른 컴포넌트가 포함하는 경우
- 기존 RouterTest 를 변경하여 적용해볼 것임.

▪ 구현된 화면

The screenshot shows a web browser window titled 'routerest' displaying a contact details page. The URL in the address bar is 'localhost:8081/#/contacts/1003'. The page has a header with '(주)OpenSG' and a navigation menu with 'Home', 'About', and 'Contacts'. Below the menu is a section titled '연락처' containing a grid of names: 김유신, 장보고, 관창, 안중근, 강감찬, 정몽주, 이순신, 김시민, 정약용. A purple box highlights a table below this grid, which contains the following data:

일련번호	1003
이름	관창
전화	010-1212-3333
주소	황산벌

A purple arrow points to the bottom-right corner of this highlighted table.

중첩 라우트



■ src/App.vue 변경

```
<script>
import Home from './components/Home.vue';
import About from './components/About.vue';
import Contacts from './components/Contacts.vue';
import ContactByNo from './components/ContactByNo.vue';
import VueRouter from 'vue-router';

const router = new VueRouter({
  routes : [
    { path: '/', component: Home },
    { path: '/home', component: Home },
    { path: '/about', component: About },
    {
      path: '/contacts', component: Contacts,
      children : [
        { path : ':no', component : ContactByNo },
      ]
    }
  ]
})
export default {
  name : 'app',
  router
}
</script>
```



■ 예제 12-09 : src/components/Contacts.vue

```
<template>
<div>
  <h1>연락처</h1>
  <div class="wrapper">
    <div class="box" v-for="c in contacts" :key="c.no">
      <router-link v-bind:to="/contacts/" +c.no>{{c.name}}</router-link>
    </div>
  </div>
  <router-view></router-view>
</div>
</template>
<script>
.....(변경사항 없음)
</script>
<style>
.....(변경사항 없음)
</style>
```



■ 예제 12-10 : src/components/ContacByNo.vue 변경

```
<template>
  <div>
    <hr class="divider"></hr>
    <table class="detail table table-bordered">
      .....(생략)
    </table>
  </div>
</template>

<script>
import contactlist from '../ContactList';

export default {
  .....(생략)
  created : function() {
    this.no = this.$route.params.no;
  },
  .....(생략)
}
</script>

<style>
.divider { height: 3px; margin-left: auto; margin-right: auto;
  background-color:#FF0066; color:#FF0066; border: 0 none; }
table.detail { width:400px; }
</style>
```



■ 여기까지의 코드로 실행하면 약간의 문제 발생

- 연락처 명단을 처음 클릭할 때는 잘 실행됨
 - created 이벤트 혹은 실행
 - this.\$route.params.no 값 획득
- 계속해서 명단을 클릭하면...
 - created 이벤트 혹은 실행 안됨.
 - 이유는 이미 화면에 ContactByNo.vue 컴포넌트 인스턴스가 생성되어 있으므로 created 이벤트 혹은 실행되지 않음
 - 이 때문에 \$route 속성에 대해 관찰 속성을 적용해야 함.

```
import contactlist from './ContactList';
export default {
    .....
    watch : {
        //from:이전 라우트 객체, to:현재 라우트 객체
        '$route'(to, from) {
            this.no = to.params.no;
        }
    },
    .....
}
```

중첩 라우트



실행 결과

The screenshot shows a browser window displaying a Vue.js application. The application has a header with 'Home', 'About', and 'Contacts' links. The 'Contacts' link is highlighted in yellow. Below the header, there is a section titled '연락처' (Contacts) containing several buttons with names: 김유신, 정보고, 관창, 안중근, 강감찬, 정몽주, 이순신, 김시민, and 정약용. To the right of this is a table with details for contact '1009': 일련번호 (1009), 이름 (정약용), 전화 (010-1212-3339), and 주소 (남양주). The developer tools (Elements tab) are open, showing the component hierarchy. A purple arrow points from the browser's DOM structure to the 'Contactbyno' component in the developer tools. The developer tools also show the 'data' and 'computed' properties for the 'Contactbyno' component.

Ready. Detected Vue 2.3.4.

Components Vuex Events Refresh

Filter components

<Contactbyno> Inspect DOM Filter inspected data

data

- \$route: Object
- contacts: Array[9]
- no: "1009"

computed

- contact: Object
 - address: "남양주"
 - name: "정약용"
 - no: 1009
 - tel: "010-1212-3339"

router-view: /contacts

router-view: /contacts/:no



■ Named Route

- 라우트 정보에 고유 이름을 부여하고 이 이름을 이용해 내비게이션!!
- 복잡한 URI 경로 대신에 좀더 간단한 고유 이름을 사용할 수 있음.
- URI 경로가 변경되더라도 컴포넌트에서의 내비게이션 정보는 변경(X)

■ 이전 routertest 프로젝트 수정

명명된 라우트



■ 예제 12-12 : src/App.vue

- route 객체에 이름 부여

```
<script>
import Home from './components/Home.vue';
import About from './components/About.vue';
import Contacts from './components/Contacts.vue';
import ContactByNo from './components/ContactByNo.vue';
import VueRouter from 'vue-router';

const router = new VueRouter({
  routes : [
    { path: '/', component: Home },
    { path: '/home', name:'home', component: Home },
    { path: '/about', name:'about',component: About },
    { path: '/contacts', name:'contacts', component: Contacts,
      children : [
        { path : ':no', name:'contactbyno', component : ContactByNo },
      ]
    }
  ]
})

export default {
  router
}
</script>
```

명명된 라우트



■ 예제 12-13 : src/App.vue의 router-link 변경

```
<template>
<div>
.....
<ul>
<li>
  <router-link v-bind:to="{ name:'home' }">Home</router-link>
</li>
<li>
  <router-link v-bind:to="{ name:'about' }">About</router-link>
</li>
<li>
  <router-link v-bind:to="{ name:'contacts' }">Contacts</router-link>
</li>
</ul>
.....
</div>
</template>
.....
```

명명된 라우트



■ route name과 함께 query, params를 전달하고 싶다면?

표 12-01

전체 URI 경로 형태	Router-link 예
/contacts/no	<pre><router-link v-bind:to="{ name:'contacts', params: { no:1003 } }"> </router-link> ex) /contacts/1003</pre>
/contacts?pageno=2	<pre><router-link v-bind:to="{ name:'contacts', query: { pageno:2 } }"> </router-link> ex) /contacts?pageno=2</pre>

명명된 라우트



■ 예제 12-14 : src/components/Contacts.vue 변경

```
<template>
  <div>
    <h1>연락처</h1>
    <div class="wrapper">
      <div class="box" v-for="c in contacts" :key="c.no">
        <router-link v-bind:to="{ name:'contactbyno', params:{ no:c.no } }">
          {{c.name}}
        </router-link>
      </div>
    </div>
    <router-view></router-view>
  </div>
</template>
```





■ 이제까지의 내비게이션

- <router-link>를 이용한 선언적 내비게이션

■ 라우터 객체의 push 메서드

- this.\$router.push() 메서드

- push(location [, completeCallback] [, abortCallback])

- 사용 방법

- 문자열 직접 전달 : this.\$router.push('/home')

- 객체 정보로 전달 : this.\$router.push({ path: '/about' })

- 명명된 라우트 사용 : this.\$router.push({ name: 'contacts', params: { no: 1002 } })

- 쿼리 문자열 전달 ex) /contacts?pageno=1&pagesize=5

- : router.push({ path: '/contacts', query: { pageno: 1, pagesize: 5 } })

프로그래밍 방식의 라우팅 제어



- 이전 routertest 프로젝트에 프로그래밍 방식 라우팅 추가
- 예제 12-15 : src/components/Contacts.vue

```
<template>
    ....(생략)
    <div class="wrapper">
        <div class="box" v-for="c in contacts" :key="c.no">
            <span @click="navigate(c.no)" style='cursor:pointer'>[ {{c.name}} ]</span>
        </div>
    </div>
    <router-view></router-view>
</div>
</template>
<script>
.....
export default {
    ....(생략)
    methods : {
        navigate(no) {
            if (confirm("상세 정보를 보시겠습니까?")) {
                this.$router.push({ name: 'contactbyno', params: { no: no }}, function() {
                    console.log("/contacts/" + no + "로 이동 완료!")
                })
            }
        }
    }
}
</script>
```

프로그래밍 방식의 라우팅 제어



■ 실행 결과

The screenshot shows a browser window titled 'routerest' displaying a contact details page for '관창' (Gwangchang) with ID 1003. The contact information includes: 일련번호 (1003), 이름 (관창), 전화 (010-1212-3333), and 주소 (황산벌). Below the contact details, there is a row of buttons: [김유신], [장보고], [관창], [안중근], [강감찬], [정몽주], [이순신], [김시민], and [정약용]. A purple arrow points from the browser's address bar ('localhost:8081/#/contacts/1003') to a modal dialog box. The dialog box has the title 'localhost:8081 내용:' and the message '상세 정보를 보시겠습니까?'. It contains two buttons: '확인' (Confirm) and '취소' (Cancel). Another purple arrow points from the bottom right of the dialog box to the developer tools' console tab.

localhost:8081 내용:
상세 정보를 보시겠습니까?

확인 취소

localhost:8081 내동!
/contacts/1005 로 이동 완료!
/contacts/1006 로 이동 완료!
/contacts/1004 로 이동 완료!
/contacts/1003 로 이동 완료!



■ 내비게이션 보호 : Navigation Guards

- 기능
 - 다른 경로로 리디렉션
 - 때로는 내비게이션을 취소
 - 때로는 내비게이션을 보호
- 적용 수준
 - 전역 수준, 라우트 정보 수준, 컴포넌트 수준

프로그래밍 방식의 라우팅 제어



■ 전역 수준 적용

```
const router = new VueRouter({ ... })  
  
router.beforeEach((to, from, next) => {  
    // ...  
})  
router.afterEach((to, from) => {  
    // ...  
})
```

- beforeEach : 라우팅이 일어나기 전
- afterEach : 라우팅이 일어난 후
 - to : 이동하려는 대상 Route 객체
 - from : 이동하기 전의 Route 객체
 - next : 함수!! next를 이용하여 경로 이동에 대한 상세 작업 수행. 라우팅이 일어나기 전에만 사용 가능
- next 함수 이용
 - next(): 다음 이벤트 흡으로 이동시킵니다. 이 함수를 호출하지 않으면 다음 이벤트 흡으로 이동하지 않습니다.
 - next(false): 현재 네비게이션을 중단합니다. from 라우트 객체의 URL로 재설정됩니다.
 - next(경로): 지정된 경로 리디렉션합니다. 현재의 네비게이션이 중단되고 새로운 네비게이션이 시작됩니다.
 - next(error): next에 전달된 인자가 Error객체라면 네비게이션이 중단되고 router.onError()를 이용해 등록 된 콜백에 에러가 전달됩니다.

프로그래밍 방식의 라우팅 제어



■ 라우트별 보호 기능

```
const router = new VueRouter({
  routes: [
    {
      path: '/contacts/:no',
      component: ContactByNo,
      beforeEnter: (to, from, next) => {
        //.....
      }
    }
  ]
})
```

- URL 경로 별로 라우트 정보에 beforeEnter 메서드를 작성함.
- to, from, next 인자는 전역 수준과 동일함.

프로그래밍 방식의 라우팅 제어



- 컴포넌트 수준의 내비게이션 보호 기능

```
const Foo = {  
  template: `...`,  
  beforeRouteEnter (to, from, next) {  
  
    },  
  beforeRouteLeave (to, from, next) {  
  
    },  
  beforeRouteUpdate(to, from, next) {  
  
    }  
}
```

- beforeRouteEnter : 렌더링하는 라우트 이전에 호출되는 흑. 이 흑이 호출되는 시점에는 Vue 인스턴스가 만들어지지 않았기 때문에 this를 이용할 수 없음.
- beforeRouteLeave : 현재 경로에서 다른 경로로 빠져나갈 때 호출되는 흑.
- beforeRouteUpdate : 이미 렌더링된 컴포넌트의 경로가 변경될 때 호출되는 흑. 이미 현재 컴포넌트의 Vue 인스턴스가 만들어져 있어서 재사용될 때는 beforeRouteEnter 흑은 호출되지 않고 beforeRouteUpdate 흑이 호출됨. 이 흑 대신에 \$route 옵션에 대해 관찰속성(watched property)를 사용할 수 있음.

프로그래밍 방식의 라우팅 제어



- beforeRouteEnter 에서 Vue 인스턴스를 이용하고 싶다면 콜백 함수를 이용해 비동기 처리를 해야 함. next() 함수를 이용해 콜백 함수를 전달함.

```
beforeRouteEnter (to, from, next) {  
    next((vm) => {  
        //vm이 생성된 vue 인스턴스를 참조합니다.  
    })  
}
```

■ 네비게이션 보호 기능의 실행 흐름

- 네비게이션 시작
- 비활성화된 컴포넌트가 있다면 보호 기능 호출
- 전역수준의 beforeEach 호출
- 재사용되는 컴포넌트의 beforeRouteUpdate 혹은 호출
- 라우트정보 수준의 beforeEnter 호출
- 활성화된 컴포넌트에서 beforeRouteEnter 혹은 호출
- 네비게이션 완료.
- 전역 afterEach 혹은 호출
- DOM 갱신 트리거 됨
- 인스턴스들의 beforeRouteEnter 혹은에서 next에 전달된 콜백 호출(콜백 사용시에만)

프로그래밍 방식의 라우팅 제어



■ 내비게이션 보호 기능 이용 예제

- /contacts에서만 /contacts/:no로의 이동만 허용하도록...
- ContactByNo.vue 컴포넌트에서 관찰 속성을 사용했던 부분을 beforeRouteUpdate 흑을 사용하도록 변경
- 일단 실행 여부 확인

```
<script>
import contactlist from '../ContactList';

export default {
    .....
    //watch : {
    //  '$route' : function(to, from) {
    //    this.no = to.params.no;
    //  }
    //},
    beforeRouteUpdate(to,from,next) {
        console.log("** beforeRouteUpdate")
        this.no = to.params.no;
        next()
    },
    .....
}
</script>
```

프로그래밍 방식의 라우팅 제어



■ 예제 12-17 : src/App.vue 변경

```
const router = new VueRouter({
  routes : [
    { path: '/', component: Home },
    { path: '/home', name:'home', component: Home },
    { path: '/about', name:'about',component: About },
    {
      path:'/contacts', name:'contacts', component: Contacts,
      children : [
        {
          path : ':no', name:'contactbyno', component : ContactByNo,
          beforeEnter : (to,from,next)=> {
            console.log("@@ beforeEnter! : " + from.path + "-->" + to.path)
            if (from.path.startsWith("/contacts"))
              next();
            else
              next("/home");
          }
        },
      ]
    }
  ]
})
```

```
router.beforeEach((to, from, next) => {
  console.log("** beforeEach!!")
  next();
})
router.afterEach((to, from) => {
  console.log("** afterEach!!")
})
```

프로그래밍 방식의 라우팅 제어



■ 실행 결과

The screenshot shows two browser windows side-by-side, both titled "routerTest".
The left window displays a contact form for "이승신" (ID: 1007) with fields for Name, Phone, and Address.
The right window shows the same contact information.
Both windows have their developer tools open to the "Console" tab, which logs the following Vue.js events:

Top Tab (Contact ID 1007):

```
** beforeEach!!  
** beforeRouteUpdate  
** afterEach!!  
/contacts/1006 로 이동 완료!  
** beforeEach!!  
** beforeRouteUpdate  
** afterEach!!  
/contacts/1007 로 이동 완료!
```

Bottom Tab (Home page):

```
** beforeEach!!  
** afterEach!!  
** beforeEach!!  
@@ beforeEnter! : /about-->/contacts/1004  
** beforeEach!!  
** afterEach!!
```

A purple box highlights the log entries for the top tab, and a purple arrow points from it to the bottom tab's log entry for the same event.

프로그래밍 방식의 라우팅 제어



- 내비게이션 보호 기능의 적용 분야
 - 애플리케이션의 인증 처리
 - 전역 내비게이션 보호 메서드인 `beforeEach()`에서 사용자의 인증 여부 확인
 - 인증되지 않았거나 접근 권한이 없다면 `next()` 메서드를 이용해 로그인 화면으로 강제이동!!

라우팅 모드



■ vue-router 객체의 기본 설정 모드는 해시모드(hash mode)

- URL에서 해시(#) 기호 다음의 경로는 페이지 내부의 이름으로 여겨짐
- 해시 이후의 경로가 변경되더라도 페이지가 다시 로드되지 않음
- 개발자 도구의 네트워크 탭으로 확인

The screenshot shows a browser window with the title "routerest" and the URL "localhost:8080/#/about". The page content displays "About" and red Korean text: "URL을 직접 변경해도 새로운 요청이 서버로 전송되지 않습니다." (Changing the URL directly does not send a new request to the server). The browser's developer tools Network tab is open, with a purple box highlighting the "Network" tab itself. A purple arrow points from the text "해시모드" in the slide to the "Network" tab. Another purple arrow points from the text "페이지가 다시 로드되지 않음" to the "Preserve log" checkbox in the Network tab settings.

URL을 직접 변경해봅니다.

(주) OpenSG

About

URL을 직접 변경해도 새로운 요청이 서버로 전송되지 않습니다.

Recording network activity...

Perform a request or hit F5 to record the reload.

라우팅 모드



■ 해시를 사용하지 않으면

- vue-router 객체의 mode를 history로 지정

```
<script>
.....
const router = new VueRouter({
  mode : 'history',
  routes : [
    .....
  ]
})
.....
</script>
```

URL을 직접 변경해봅니다.

(주) OpenSG

About

URL을 직접 변경하면 새로운 요청이 서버로 전송된 것을 알 수 있습니다.

Name	Status	Type	Initiator	Size	Time	Waterfall
websocket	101	websock...	websocket.js?0f...	0 B	Pending	
about	304	docum...	Other	241 B	303 ms	
build.js	304	script	about	181 B	11 ms	
bootstrap.css	200	styleshe...	addSty http://localhost:8080/about	ms		
info?t=1499904160726	200	xhr	abstract-xhr.js?	368 B	7 ms	

5 requests | 790 B transferred | Finish: 882 ms | DOMContentLoaded: 847 ms | Load: 861 ms

라우팅 모드



■ 잘못된 경로 요청시 오류 메시지를 나타내려면...

- catch-all 라우트와 오류메시지를 보여줄 컴포넌트를 추가
- 예제 12-20 : src/components/NotFound.vue

```
<template>
  <h1>요청하신 경로는 존재하지 않습니다</h1>
</template>
```

- 예제 12-21 : src/App.vue 변경

```
const router = new VueRouter({
  mode : 'history',
  routes : [
    .....
    { path: '*', component: NotFound }
  ]
})
```

라우팅 모드



실행 결과

The screenshot shows a browser window titled "routerest" displaying a 404 error page. The URL in the address bar is "localhost:8080/blahblah". The page content is "(주) OpenSG" with a purple header bar containing "Home", "About", and "Contacts" links. Below the header, the text "요청하신 경로는 존재하지 않습니다" is displayed. To the right, the Chrome DevTools sidebar is open, specifically the "Vue" tab. It shows a component tree under "Root": <Root> -> <App> -> <RouterLink> -> <RouterLink> (highlighted in blue) -> <RouterLink> -> <NotFound> (highlighted in orange). The "router-view: *" prop is also visible. A tooltip on the right side of the sidebar says "Select a component instance to inspect." A purple arrow points from the browser's address bar to the DevTools sidebar.

라우팅 정보를 속성으로 연결하기



■ 컴포넌트가 라우트 객체에 의존적인 것은?

- Vue 컴포넌트에서 `this.$route` 객체의 정보에 의존적인 것은 재사용성 측면에서 바람직하지 않음.
- 라우트 객체의 `params` 속성 정보를 컴포넌트의 `props`로 전달할 수 있음
 - `props` 옵션을 `true`로 지정 → `route.params` 정보를 동일한 `props`로 전달
 - `created` 이벤트 혹은 필요하지 않음.
 - `beforeRouteUpdate` 내비게이션 보호 기능도 필요하지 않음

```
const router = new VueRouter({  
  mode : 'history',  
  routes : [  
    .....  
    {  
      path:'/contacts', name:'contacts', component: Contacts,  
      children : [  
        { path : ':no', name:'contactbyno', component : ContactByNo, props:true }  
      ]  
    },  
    { path: '*', component: NotFound }  
  ]  
})
```

라우팅 정보를 속성으로 연결하기



■ params가 아닌 query 등이 속성에 부여되어야 한다면?

- props 옵션에 함수를 지정
- 함수가 리턴하는 값이 props로 주어짐

```
//no속성, path 속성에 라우트 정보를 부여함.
function connectQueryToProp(route) {
  return { no : route.query.no, path: route.path };
}

const router = new VueRouter({
  routes: [
    .....
    { path: '/contactbyno', component: ContactByNo, props: connectQueryToProp }
  ]
})
```

연락처 애플리케이션에 라우팅 기능 적용



■ 11장까지의 예제

- Vuex, Dynamic Component, axios 사용

■ 기존 예제를 vue-router를 사용하는 앱으로 변경

■ 라우팅 경로 정보 결정

- Home.vue, About.vue 추가
- 연락처 입력, 수정을 구분하기 위해 URI 경로 정보를 활용
- '이름' 필드는 명명된 라우트를 사용하기 위한 고유 이름

표 12-02 라우팅 경로 정보

URI 경로	이름	컴포넌트
/		Home.vue 컴포넌트(/home으로 리디렉션)
/home	home	Home.vue 컴포넌트
/about	about	About.vue 컴포넌트
/contacts	contacts	ContactList.vue 컴포넌트
/contacts/add	addcontact	ContactForm 컴포넌트(입력)
/contacts/update/:no	updatecontact	ContactForm 컴포넌트(수정)
/contacts/photo/:no	updatephoto	UpdatePhoto 컴포넌트(사진 변경)

연락처 애플리케이션에 라우팅 기능 적용



■ 초기 설정 작업

- 예제 12-24 : Home.vue, About.vue 추가

```
<template>
  <!-- About.vue도 Home.vue와 동일하게 작성합니다.-->
  <div id="example">
    <div class="panel panel-default">
      <div class="panel-heading">Home</div>
      <div class="panel-body">Home 화면입니다</div>
    </div>
  </div>
</template>
<style>
#example { margin:10px auto; max-width: 820px; min-width: 820px;
  padding:0px; position:relative; font: 13px "verdana"; }
</style>
```

- npm install --save vue-router

연락처 애플리케이션에 라우팅 기능 적용



■ Vuex 상태 관리 기능 변경

■ 변경사항

- 기존 예제는 Dynamic Component를 이용했으므로 mode, currentView와 같은 state가 필요했음
- 하지만 vue-router를 적용하면 더이상 필요하지 않음
- 두가지 상태 속성(mode, currentView)이 필요없어지면서 관련된 변이와 액션도 필요 없음

■ 예제 12-25 : src/constant.js(상수 변경)

```
export default {  
  FETCH_CONTACTS : "fetchContacts",          //연락처 조회  
  ADD_CONTACT : "addContact",                //연락처 추가  
  UPDATE_CONTACT : "updateContact",          //연락처 수정  
  UPDATE_PHOTO : "updatePhoto",               //사진 수정  
  DELETE_CONTACT : "deleteContact",          //연락처 삭제  
  FETCH_CONTACT_ONE : "fetchContactOne",     //연락처 한건 조회  
  INITIALIZE_CONTACT_ONE : "initializeContactOne" //연락처 한건 초기화(빈문자열로)  
}
```

연락처 애플리케이션에 라우팅 기능 적용



■ 예제 12-26 : src/store/state.js

```
import Constant from '../constant';
import CONF from '../Config.js';
export default {
    contact : { no:0, name:"", tel:"", address:"", photo:@"" },
    contactlist : {
        pageno:1, pagesize: CONF.PAGESIZE, totalcount:0, contacts:[]
    }
}
```

■ 예제 12-27 : src/store/mutations.js

```
import Constant from '../constant';

//상태를 변경하는 기능만을 뽑아서...
export default [
    [Constant.FETCH_CONTACTS] : (state, payload) => {
        state.contactlist = payload.contactlist;
    },
    [Constant.FETCH_CONTACT_ONE] : (state, payload) => {
        state.contact = payload.contact;
    },
    [Constant.INITIALIZE_CONTACT_ONE] : (state) => {
        state.contact = { no:"", name:"", tel:"", address:"", photo:@"" };
    }
]
```

연락처 애플리케이션에 라우팅 기능 적용



- 상태와 변이에서의 변경 사항
 - mode, currentView 상태 제거
 - mode, currentView 상태와 관련된 변이 제거
 - FETCH_CONTACT_ONE 변이 추가 : 한건의 연락처 조회 기능(수정을 위해서)
 - INITIALIZE_CONTACT_ONE 변이 추가 : 연락처 한건 정보 빈문자열로 초기화
- 예제 12-28 : src/store/actions.js
 - 코드 생략
 - mode, currentView 상태와 관련된 액션 제거
 - 더이상 화면 전환을 위해 mode, currentView 상태를 이용하지 않기 때문에...
 - 연락처 추가, 수정, 사진 변경 액션에서도 CANCEL_FORM 액션을 일으킬 필요가 없음
 - router 객체의 push 메서드로 URL 경로를 이동시키면 화면이 전환되기 때문에...
 - FETCH_CONTACT_ONE, INITIALIZE_CONTACT_ONE 액션 추가

연락처 애플리케이션에 라우팅 기능 적용



■ main.js에 라우팅 기능 추가

- 예제 12-29 : src/main.js 변경

```
import Vue from 'vue'  
import App from './App'  
import store from './store'  
import VueRouter from 'vue-router';  
  
import Home from './components/Home';  
import About from './components/About';  
import ContactList from './components/ContactList';  
import ContactForm from './components/ContactForm';  
import UpdatePhoto from './components/UpdatePhoto';  
  
Vue.use(VueRouter);  
Vue.config.productionTip = false  
  
const router = new VueRouter({  
  routes: [  
    { path: '/', redirect: '/home' },  
    { path: '/home', name: 'home', component: Home },  
    { path: '/about', name: 'about', component: About },
```

```
  {  
    path: '/contacts', name: 'contacts',  
    component: ContactList,  
    children: [  
      { path: 'add', name: 'addcontact',  
        component: ContactForm },  
      { path: 'update/:no', name: 'updatecontact',  
        component: ContactForm, props: true },  
      { path: 'photo/:no', name: 'updatephoto',  
        component: UpdatePhoto, props: true }  
    ]  
  },  
]  
})  
  
new Vue({  
  store,  
  router,  
  el: '#app',  
  template: '<App/>',  
  components: { App }  
})
```

연락처 애플리케이션에 라우팅 기능 적용



■ main.js 코드 검토

- 20~26행과 같이 중첩 라우트 사용
- 명명된 라우트 사용. 고유 명칭 사용. 표 12-02의 내용을 적용
- 17행 redirect 다른 경로로 이동시키는 역할 수행 (/ → /home)
- props : true 옵션 적용. URI 경로 상의 :no 값은 해당 컴포넌트의 props로 전달됨

연락처 애플리케이션에 라우팅 기능 적용



- 예제 12-30 : src/App.vue 변경

```
<template>
  <div id="container">
    <div class="page-header">
      <h1 class="text-center">연락처 관리 애플리케이션</h1>
      <p>(Vue-router + Vuex + Axios) </p>
      <div class="btn-group">
        <router-link to="/home" class="btn btn-info menu">Home</router-link>
        <router-link to="/about" class="btn btn-info menu">About</router-link>
        <router-link to="/contacts" class="btn btn-info menu">Contacts</router-link>
      </div>
    </div>
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: 'app'
}
</script>

<style scoped>
.....(생략)
</style>
```

연락처 애플리케이션에 라우팅 기능 적용



■ ContactList.vue 컴포넌트 변경

- 예제 12-31 : src/components/ContactList.vue
 - template

```
<template>
  <div>
    <p class="addnew">
      <router-link class="btn btn-primary" v-bind:to="{ name:'addcontact' }">
        새로운 연락처 추가하기</router-link>
    </p>
    <div id="example">
      <table id="list" class="table table-striped table-bordered table-hover">
        .....(생략)
      </table>
    </div>
    <paginate ref="pagebuttons"
      :page-count="totalpage" :page-range="7"
      :margin-pages="3" :click-handler="pageChanged"
      :prev-text="'이전'" :next-text="'다음'"
      :container-class="'pagination'"
      :page-class="'page-item'">
    </paginate>
    <router-view></router-view>
  </div>
</template>
```

연락처 애플리케이션에 라우팅 기능 적용



- 예제 12-31 : src/components/ContactList.vue

- script

```
<script>
import Constant from '../constant';
import { mapState } from 'vuex';
import Paginate from 'vuejs-paginate';
import _ from 'lodash';

export default {
  name : 'contactList',
  components : { Paginate },
  computed : _.extend(
    {
      totalpage : function() {
        var totalcount = this.contactlist.totalcount;
        var pagesize = this.contactlist.pagesize;
        return Math.floor((totalcount - 1) / pagesize) + 1;
      }
    },
    mapState([ 'contactlist' ])
  ),
}
```

(다음 페이지에 이어짐)

연락처 애플리케이션에 라우팅 기능 적용



- 예제 12-31 : src/components/ContactList.vue

- script

```
mounted : function() {
    var page = 1;
    if (this.$route.query && this.$route.query.page) {
        page = parseInt(this.$route.query.page);
    }
    this.$store.dispatch(Constant.FETCH_CONTACTS, { pageno:page });
    this.$refs.pagebuttons.selected = page-1;
},
watch : {
    '$route' : function(to, from) {
        if (to.query.page && to.query.page != this.contactlist.pageno) {
            var page = to.query.page;
            this.$store.dispatch(Constant.FETCH_CONTACTS, { pageno:page });
            this.$refs.pagebuttons.selected = page-1;
        }
    }
},
methods : {
    pageChanged : function(page) {
        this.$router.push({ name: 'contacts', query: { page: page } })
    },
}
```

(다음 페이지에 이어짐)

연락처 애플리케이션에 라우팅 기능 적용



- 예제 12-31 : src/components/ContactList.vue

- script

```
editContact : function(no) {
    this.$router.push({ name: 'updatecontact', params : { no:no } })
},
deleteContact : function(no) {
    if (confirm("정말로 삭제하시겠습니까?") == true) {
        this.$store.dispatch(Constant.DELETE_CONTACT, {no:no});
        this.$router.push({ name: 'contacts' })
    }
},
editPhoto : function(no) {
    this.$router.push({ name: 'updatephoto', params: { no: no } })
}
}
</script>
```

연락처 애플리케이션에 라우팅 기능 적용



■ ContactList.vue 코드 검토

- 32~41행의 페이징 컴포넌트 : App.vue에 있던 것을 ContactList.vue로 이동시킴
- 연락처 추가 버튼 : <router-link>로 처리. 명명된 라우트 이용
- 65행 mounted 이벤트 혹은
 - ContactList.vue 컴포넌트가 마운트되어 활성화될 때 실행됨.
 - \$route.query를 이용해 page 값을 읽어냄 --> 특정 페이지 조회하도록 초기화
- 73행 \$route에 대한 관찰 속성
 - 컴포넌트 마운트 후 라우트 경로만 바뀌는 동안에 페이지를 변경할 수 있도록 처리함.
 - 관찰 속성을 사용한 코드는 내비게이션 보호 기능의 beforeRouteUpdate 혹은을 이용하도록 변경할 수 있음.

```
beforeRouteUpdate(to,from,next) {
    if (to.query.page && to.query.page != this.contactlist.pageno) {
        var page = to.query.page;
        this.$store.dispatch(Constant.FETCH_CONTACTS, { pageno:page });
        this.$refs.pagebuttons.selected = page-1;
        next();
    }
},
```

연락처 애플리케이션에 라우팅 기능 적용



■ ContactForm.vue, UpdatePhoto.vue 컴포넌트 수정

- 예제 12-32 : src/components/ContactForm.vue 변경
 - props로 전달받던 mode를 컴포넌트 지역 data 옵션으로 처리
 - 로컬 data 옵션값인 mode 기본값을 add로 설정하고 URI 경로 상에 /add, /update 문자열의 포함 여부를 확인해 mode 값을 설정
 - mode 로컬 data 옵션 값에 따라 추가 화면, 수정 화면을 보여줄지를 결정함.
 - props를 이용해 no 값을 전달받음
 - 예제 12-29 src/main.js에서 props:true를 지정했으므로 /contacts/update/:no 경로의 :no 위치의 값이 no props 값으로 전달됨.
 - 수정, 추가, 취소 버튼을 클릭하면 연락처 화면으로 돌아가기 위해 this.\$router.push() 메서드를 이용해 내비게이션!!
- 예제 12-33 : src/components/UpdatePhoto.vue
 - ContactForm.vue와 유사함.

연락처 애플리케이션에 라우팅 기능 적용



■ 실행 결과1

The screenshot shows a browser window for a contact management application. The title bar says "contactsapp" and the address bar shows "localhost:8080/#/contacts?page=8". The main content area displays a table of contacts:

이름	전화번호	주소	사진	편집/삭제
Sophie Torres	010-3456-8263	서울시		<button>편집</button> <button>삭제</button>
Rosebud Smith	010-3456-8262	서울시		<button>편집</button> <button>삭제</button>
Rhu Clark	010-3456-8261	서울시		<button>편집</button> <button>삭제</button>
Sean Adams	010-3456-8260	서울시		<button>편집</button> <button>삭제</button>
Gladys Young	010-3456-8259	서울시		<button>편집</button> <button>삭제</button>

Below the table is a navigation bar with buttons for "이전", page numbers (1, 2, 3, ..., 5, 6, 7, 8, 9, 10, 11, ..., 18, 19, 20), and "다음".

The right side of the screenshot shows the developer tools' element inspector. It identifies the Vue.js version as "Ready. Detected Vue 2.3.4". The component tree shows the following structure:

- <Root>
- <App>
 - <RouterLink>
 - <RouterLink>
 - <RouterLink>
 - <ContactList> **router-view: /contact**
 - <RouterLink>
 - <Paginate>

A purple box highlights the "router-view: /contact" part of the ContactList component. A purple arrow points from this box to a text overlay that says "Select a component instance to inspect.".

연락처 애플리케이션에 라우팅 기능 적용



■ 실행 결과2

The screenshot shows a browser window displaying the "연락처 관리 애플리케이션" (Contact Management Application). The page lists contacts and allows users to change their profile picture. On the right, the Vue DevTools extension is open, showing the component tree for the application. A purple box highlights the `<ContactList>` component, which contains a `router-view` directive with the value `/contact`. An arrow points from this highlighted area to a tooltip that says "Select a component instance to inspect".

연락처 관리 애플리케이션

:: 사진 변경

현재 사진

새로운 연락처 추가하기

이름	선택된 파일 없음	편집	삭제		
Sophie Torres	<input type="file"/>	편집	삭제		
Rosebud Smith	<input type="file"/>	편집	삭제		
Rhu Clark	<input type="file"/>	편집	삭제		
Sean Adams	010-3456-8260	서울시	<input type="file"/>	편집	삭제
Gladys Young	010-3456-8259	서울시	<input type="file"/>	편집	삭제

이전 1 2 3 ... 5 6 7 8 9 10 11 ... 18 19 20 다음

Ready. Detected Vue 2.3.4.

Filter components

<Root>

<App>

- <RouterLink>
- <RouterLink>
- <RouterLink>
- <ContactList> router-view: /contact
- <UpdatePhoto> router-view: /contact
- <RouterLink>
- <Paginate>

Select a component instance to inspect.



■ API 호출 지연 시간 발생

- API 호출시 지연시간이 발생하면 화면 전환이 부드럽지 않음
- src/Config.js 파일 수정
 - /contacts_long : 1초의 의도적 지연시간 발생

```
var BASE_URL = "/api";  
  
export default {  
    PAGESIZE : 5,  
  
    //전체 연락처 데이터 요청(페이지 포함)  
    FETCH : BASE_URL + "/contacts_long",  
    .....(생략)  
}
```

- 지연 시간의 발생동안 처리 중임을 알릴 수 있는 UI가 제공되어야 함.
 - spinner UI



■ 스피너 컴포넌트 작성

- vue-simple-spinner 컴포넌트 사용
 - <https://github.com/dzwillia/vue-simple-spinner>
- npm install --save vue-simple-spinner
- 예제 12-35 : src/components>Loading.vue
 - 스피너 UI를 보여주기 위한 컴포넌트
 - 단순한 UI만 제공



Loading...



■ 상태와 변이, 액션 변경

- Loading.vue를 보여줄지를 결정하기 위한 vuex의 상태를 추가
- 이것을 변경하는 변이와 관련된 액션 추가
- 예제 12-36 : src/constant.js에 상수 추가
 - CHANGE_ISLOADING
- 예제 12-37 : src/store/state.js (상태 변경)

```
import Constant from './constant';
import CONF from './Config.js';

export default {
  isloading : false,
  contact : { no:0, name:"", tel:"", address:"", photo:@"" },
  contactlist : {
    pageno:1, pagesize: CONF.PAGESIZE, totalcount:0, contacts:[]
  }
}
```

지연 시간에 대한 처리



- 예제 12-38 : src/store/mutations.js (변이 변경)

```
import Constant from '../constant';
export default {

    .....
    [Constant.CHANGE_ISLOADING] : (state, payload) => {
        state.isloading = payload.isloading;
    }
}
```

- 예제 12-39 : src/store/actions.js (액션 변경)

- axios를 이용해 조회, 추가, 수정, 삭제하기 전 스피너가 나타나고 응답 수신 후 스피너가 사라짐
- 추가, 수정, 삭제 시에는 작업 후 연락처 데이터를 다시 조회하므로 FETCH_CONTACTS 액션에서 사라지도록 처리함.

```
[Constant.ADD_CONTACT] : (store) => {
    store.dispatch(Constant.CHANGE_ISLOADING, { isloading:true });
    contactAPI.addContact(store.state.contact)
    .then((response)=> {
        if (response.data.status == "success") {
            store.dispatch(Constant.FETCH_CONTACTS, { pageno: 1});
        } else {
            console.log("연락처 추가 실패 : " + response.data);
        }
    })
},
```

지연 시간에 대한 처리



- 예제 12-40 : src/App.vue
 - Loading.vue 컴포넌트를 App.vue에 추가

```
<template>
<div id="container">
  <div class="page-header">
    <h1 class="text-center">연락처 관리 애플리케이션</h1>
    <p>(Vue-router + Vuex + Axios)</p>
    <div class="btn-group">
      <router-link to="/home" class="btn btn-info menu">Home</router-link>
      <router-link to="/about" class="btn btn-info menu">About</router-link>
      <router-link to="/contacts" class="btn btn-info menu">Contacts</router-link>
    </div>
  </div>
  <router-view></router-view>
  <loading v-show="isloading"></loading>
</div>
</template>
<script>
import Loading from './components>Loading';
import { mapState } from 'vuex';
export default {
  name: 'app',
  components : { Loading },
  computed : mapState([ 'isloading' ])
}
</script>
.....
```

지연 시간에 대한 처리



■ 실행 결과

연락처 관리 애플리케이션
(Vue-router + Vuex + Axios)

새로운 연락처 추가하기

이름	전화번호	주소	사진	편집/삭제
Jennifer Harris	010-3456-8294	서울시		편집 삭제
Ambrosia Russell	010-3456-8293	서울시		편집 삭제
Isabelle Richardson	010-3456-8292	서울시		편집 삭제
Victoria Sullivan	010-3456-8291	서울시		편집 삭제
Isabella Ross	010-3456-8290	서울시		편집 삭제

이전 1 2 3 4 5 6 7 8 ... 18 19 20 다음

Vue DevTools

Ready. Detected Vue 2.3.4.

Filter components <Paginate> Filter inspected data

<App>

- <Loading> (highlighted with a purple box and arrow)
- <RouterLink>
- <RouterLink>
- <RouterLink>
- <ContactList> `router-view: /contact`
- <RouterLink>
- <Paginate> == \$vm0

data

- \$route: Object
- selected: 2

props

- clickHandler: Function
- containerClass: "pagination"
- forcePage: undefined

Console

[vue-devtools] Ready. Detected Vue v2.3.4 backend.js:1