6.Vue Router

1-1. Rouer 개념 및 설정

1. Rouer 개념

1) **라우터(Router) 개념**

- 라우터라고 하면 일반적으로 네트워크간에 데이터를 전송하는 장치.
- 뷰에서 말하는 라우터는 쉽게 말해서 URL에 따라 어떤 페이지를 보여줄지 **매핑**해주는 라이브러리

2) **뷰 라우터 (Vue Router)**

- Vue.js를 이용하여 **싱글 페이지 애플리케이션(SPA)을 구현**할 때 사용하는 Vue.js의 공식 라우터

3) **라우트 (Route)**

- 어떤 URL에 대해 어떤 페이지를 표시해야 하는지에 대한 정보

2. **라우터(router) 설정**

1) app.use(router)를 호출 함으로써 컴포넌트 내부에서 $router, $route 객체에 접근

1-2. 뷰 라우터 사용 방법

3. 뷰 라우터를 HTML과 JavaScript로 사용하는 방법

1) HTML

① **<RouterLink>**

- Vue Router 에서는 페이지를 이동할 때는 일반 a태그를 사용하는 대신 커스텀 컴포넌트인 <RouterLink>를 사용하여 다른 페이지 링크를 만들어야 함.
- 이를 통해 Vue Router는 페이지를 리로딩 하지 않고 URL에 매핑된 페이지를 렌더링 가능 .

②<**RouterView**>

- <RouterView>는 URL에 매핑된 컴포넌트를 화면에 표시

2) JavaScript

- 위에서 router를 설정할 때 **app.use(router)를 호출**.
  이렇게 호출 함으로써 모든 자식 컴포넌트에 router, route 같은 객체를 사용 가능.
  그리고 이러한 객체는 페이지 이동 또는 현재 활성 라우트(경로 매핑)정보에 접근하는 데 사용할 수 있음

① router

라우터 인스턴스로 JavaScript에서 **다른 페이지(컴포넌트)로 이동가능**

Options API : this.$router

Composition API : useRouter()

template : $router

② route

현재 **활성 라우트 정보에 접근**할 수 있음. (이 속성은 읽기 전용 입니다.)

Options API : this.$route

Composition API : useRoute()

template : $route

1-3. 동적 라우트 매칭

1. 주어진 패턴을 가진 라우트를 동일한 컴포넌트에 매핑해야하는 경우가 자주 있음.
2. 예를 들어 사용자 목록(User List)은 /users와 같은 경로에 매핑되면 되지만
3. 사용자 상세(User Detail)는 **사용자 식별자 별**로 같은 컴포넌트에 매핑
4. 예를 들어 /users/detail URL은 모두 같은 경로('/users/**:id**')에 매핑, 동적 세그먼트는 **콜론(:)으로 표시**
5. 컴포넌트에서 동적 세그먼트의 값은 $route.params 필드로 접근
6. 동일한 라우트에 여러 동적 세그먼트를 가질 수 있으며, $route.params 필드에 매핑
7. 동일 Route 예시

Path	URL Example	$route.params
/users/:username	/users/kkk	{ username: 'kkk' }
/users/:username/board/:id	/users/kkk/board/123	{ username: 'kkk', board: '123' }

8. Query , hash 예시

- $route.params 외에도 $route 객체는 $route.query(쿼리스트링), $route.hash(해시태그) 등과 같은 다른 유용한 정보도 노출

URL Example	$route
/users?searchText=love	{ params: {...}, hash: '...', query: { searchText: love } }
/users/alice#profile	{ params: {...}, hash: 'profile', query: { ... } }

1-4-1. 프로그래밍 방식 네비게이션

1. **<RouterLink>**를 사용하여 선언적 네비게이션용 anchor 태그를 사용하는 것 외에도 라우터 인스턴스 메소드를 사용하여 프로그래밍 방식으로 이를 수행 할 수 있음.

2. RouterLink 이외 방식

① router.push

다른 URL로 이동하려면 **router.push**를 사용가능 . 이 메소드는 새로운 항목을 히스토리 스택에 넣기 때문에 사용자가 브라우저의 뒤로 가기 버튼을 클릭하면 이전 URL로 이동.

이 메소드는 **<RouterLink>**를 클릭 할 때 내부적으로 호출되는 메소드이므로 <RouterLink :to="...">를 클릭하면 router.push(...)를 호출하는 것과 같음

② router.push 예시

항목	route
리터럴 문자열 경로	router.push('/users/detail')
경로가 있는 개체	router.push({ path: '/users/detail' })
이름을 가지는 라우트 결과->/user/detail	router.push({ name: 'user', params: { username: detail' } })
쿼리와 함께 사용, 결과->/register?plan=private	router.push({ path: '/register', query: { plan: 'private' } })
해시와 함께 사용, 결과->/about#team	router.push({ path: '/about', hash: '#team' })
params`는 `path`와 함께 사용불가 결과->/user	router.push({ path: '/user', params: { username } })

1-4-2. 프로그래밍 방식 네비게이션

3. router.replace
   router.push와 같은 역할을 하지만 유일한 차이는 새로운 **히스토리 항목**에 추가하지 않고 탐색.
   이름에서 알 수 있듯이 현재 항목을 대체.
   ① 선언적 방식 : <router-link :to="..." replace>
   ② 프로그래밍 방식 : router.replace(...)
   router.push 메소드에 replace: true속성을 추가하여 동일하게 동작

4. router.go(n)
 ① router.go(1) : 한 단계 앞으로 이동. history.forward() 유사
 ② router.go(-1) : 한 단계 뒤로 갑니다. history.back()와 유사 .

5. 라우트 컴포넌트에 속성 전달
 1) 컴포넌트에서 $route객체를 사용하면 특정 URL에서만 사용할 수 있게 되어 라우트와 강한 결합을 생성.
 즉 컴포넌트의 유연성이 제한.
 2) 이러한 결합이 꼭 나쁜 것은 아니지만 **props옵션**으로 이 동작을 분리 가능
 3) 컴포넌트와 라우터 속성을 분리방법.
 라우트에 의존된 컴포넌트

```
const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}
const routes = [{ path: '/user/:id', component: User }]
```

 라우트 의존도 해제

```
const User = {
  // make sure to add a prop named exactly like the route param
  props: ['id'],
  template: '<div>User {{ id }}</div>'
}
const routes = [{ path: '/user/:id', component: User, props: true }]
```

 이를 통해 어디서나 컴포넌트를 사용할 수 있으므로 컴포넌트 재사용 및 테스트하기가 더 쉽습니다.

1-4-3. 프로그래밍 방식 네비게이션

6. Boolean 모드
① props를 true로 설정하면 route.params가 컴포넌트 props로 설정.
② Named views
이름을 가지는 뷰(Named Views)가 있는 경우 각 Named Views에 대한 props 옵션 을 정의 .
③ const routes = [

```
{
  path: '/user/:id',
  components: { default: User, sidebar: Sidebar },
  props: { default: true, sidebar: false }
}
]
```

7. 객체 모드
① props가 객체일때 컴포넌트 props가 있는 그대로 설정. props가 정적일 때 유용.
② const routes = [

```
{
  path: '/promotion/from-newsletter',
  component: Promotion,
  props: { newsletterPopup: false }
}
]
```

8. 함수 모드
① props를 반환하는 함수를 만들 수 있음.
② 이를 통해 전달인자를 다른 타입으로 캐스팅하고 적정인 값을 라우트 기반 값과 결합.
③ const routes = [

```
{
  path: '/search',
  component: SearchUser,
  props: route => ({ query: route.query.q })
}
]
```