# Vue.js

## starting vue.js

- 자바스크립트 프레임워크
- https://vuejs.org/
- https://github.com/vuejs/awesome-vue
- Data Driven
- Template, Directive
- Data binding
- Nuxt.js, VuePress
- Vuex, Vue Router
- https://curated.vuejs.org/
- https://element.eleme.io/
- https://onsen.io/
- Vue Devtool @chrome store
- 가상 DOM

Vue.js 다운로드

**i**

- https://vuejs.org/js/vue.js

CDN 이용

```
<!-- development version, includes helpful console warnings -->
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
```

```
<!-- production version, optimized for size and speed -->
<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>
```

Vue.js 기본 구조

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8">
  <title>Vue.js App</title>
  <link href="main.css" rel="stylesheet">
</head>
```

```
<body>
  <div id="app">
    <!-- template 출력 -->
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
  <script src="main.js"></script>
</body>
</html>
```

```
var app = new Vue({
  el: '#app'
})
```

# 기본 기능

- 텍스트 바인딩

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

```
<p>{{ message }}</p>
```

- 반복 렌더링; v-for directive, list in data:
- 이벤트 사용; v-on directive, methods:
- 입력 양식과 동기화; v-model directive, data:
- 조건 분기; v-if directive, boolean in data:
- 트랜지션과 애니메이션; <transition>
- {디렉티브}{:매개변수}{.장식자}={값}; ex) v-bind:value.sync = “message”

# Options in Vue Objects

```
var app = new Vue({
```

```
  // el
  el: '#app',
  // data
  data: {
    message: 'Vue.js'
  },
  // computed
  computed: {
    computedMessage: function() {
      return this.message + '!'
    }
  },
  // lifecycle hooks
  created: function() {
    // something to what you do
  },
  // methods
  methods: {
    myMethod: function() {
      // something to what you do
    }
  }

})
```

| Lifecycles | |
|---|---|
| **Methods** | **Timing** |
| **beforeCreate** | 인스턴스가 생성되고, 리액티브 초기화가 일어나기 전 |
| **created** | 인스턴스가 생성되고, 리액티브 초기화가 일어난 후 |
| **beforeMount** | 인스턴스가 마운트되기 전 |
| **mounted** | 인스턴스가 마운트된 후 |
| **beforeUpdate** | 데이터가 변경되어 DOM에 적용되기 전 |
| **updated** | 데이터가 변경되어 DOM에 적용된 후 |
| **beforeDestroy** | Vue 인스턴스가 제거되기 전 |
| **destroyed** | Vue 인스턴스가 제거된 후 |
| **errorCaptured** | 임의의 자식 컴포넌트에서 오류가 발생했을 때 |

# Data Binding

- 템플릿에 사용하는 모든 데이터를 리액티브 데이터로 정의
- 리액티브 데이터 정의; 컴포넌트의 data 옵션에 정의
- 렌더링; mustache {{ 속성이름 }}
    - 표현식만 가능, 문장식은 불가능
    - 3항 연산자의 경우 산출 속성 computed 사용을 권장
    - 문자열이나 숫자를 변환할 때는 필터를 권장
    - 속성에는 사용 불가, 속성에 바인딩 하려면 v-bind 디렉티브 사용
    - data 상태 json으로 출력; <pre> {{ $data }} </pre>
- this; 콜백으로 익명 함수를 사용하거나, 다른 라이브러리와 함께 사용할 경우 this 변경되므로 사용에 주의

- 클래스 이름에 하이픈을 넣을 때는 작은 따옴표(')로 감쌈.
- 템플릿에서 조건 분기; v-if → 주석처리, v-show → 스타일로 보이지 않게만.
- v-if, v-else-if, v-else; key 설정
- 요소를 반복해서 렌더링; v-for = "<각 요소를 할당할 변수 이름> in <반복 대상 배열 또는 객체>" v-bind:key=".."
    - 값, 키, 인덱스 순
    - 반복 처리 순서; Object.keys()의 순서에 기반
    - v-for 안에 v-if
- 리스트
    - 추가; push, unshift ex) this.list.push(새로운 값)
    - 제거; splice ex) this.list.splice(index, 1)
    - push, pop, shift, unshift, splice, sort, reverse
    - Vue.set 메서드; this.$set(변경할 데이터, 인덱스 또는 키, 새로운 값)
    - this.list.filter(function(el) { ... })

| v-bind 장식자 | |
|---|---|
| **장식자** | **의미** |
| **.prop** | 속성 대신에 DOM 속성으로 바인딩, DOM 속성과 직접 바인딩 |
| **.camel** | 케밥케이스[1] 속성 이름을 카멜 케이스로 변환 |
| **.sync** | 양방 바인딩 |

- $el을 사용해 DOM을 직접 참조 가능, 라이프사이클 중 mounted 이후부터 사용가능.
- 속성 ref와 $ref를 사용해 참조
- $el과 $ref는 일시적 변경

| 템플릿 제어 디렉티브 | |
|---|---|
| **디렉티브** | **설명** |
| **v-pre** | 템플릿 컴파일 생략 |
| **v-once** | 한 번만 바인딩 |
| **v-text** | Mustache 대신 텍스트 콘텐츠로 렌더링 |
| **v-html** | HTML 태그를 그대로 렌더링 |
| **v-cloak** | 인스턴스 준비가 끝나면 제거 |

# Event Handling

- Event Handler, Handle
- v-on

| 이벤트 장식자 | |
|---|---|
| **장식자** | **설명** |
| **.stop** | event.stopPropagation()을 호출 |
| **.prevent** | event.preventDefault()를 호출 |
| **.capture** | 캡처 모드로 DOM 이벤트를 핸들 |
| **.self** | 이벤트가 해당 요소에서 직접 발생할 때만 핸들러를 호출 |
| **.native** | 컴포넌트의 루트 요소 위에 있는 네이티브 이벤트를 핸들 |
| **.once** | 한 번만 핸들 |
| **.passive** | {passive: true}로 DOM 이벤트를 핸들 |

| 클릭 이벤트 | |
|---|---|
| 장식자 | 설명 |
| **.left** | 마우스 왼쪽 버튼으로 눌렀을 때만 핸들러 호출 |
| **.right** | 마우스 오른쪽 버튼으로 눌렀을 때만 핸들러 호출 |
| **.middle** | 마우스 중간 버튼으로 눌렀을 때만 핸들러 호출 |

- 키 장식자

| 키 코드 별칭 | |
|---|---|
| 별칭 | 의미 |
| **.enter** | Enter(엔터) 키를 눌렀을 때 |
| **.tab** | Tab(탭) 키를 눌렀을 때 |
| **.delete** | Delete(딜리트) 키를 눌렀을 때 |
| **.esc** | ESC 키를 눌렀을 때 |
| **.space** | Space(스페이스) 키를 눌렀을 때 |
| **.up** | 화살표 위 키를 눌렀을 때 |
| **.down** | 화살표 아래 키를 눌렀을 때 |
| **.left** | 화살표 왼쪽 키를 눌렀을 때 |
| **.right** | 화살표 오른쪽 키를 눌렀을 때 |

| 시스템 장식자 | |
|---|---|
| 별칭 | 의미 |
| **.ctrl** | Ctrl(컨트롤) 키가 눌린 경우 |
| **.alt** | Alt(안트) 키가 눌린 경우 |
| **.shift** | Shift(시프트) 키가 눌린 경우 |
| **.meta** | Meta(메타) 키가 눌린 경우 |

- 양방향 데이터 바인딩; v-model
  - 데이터 바인딩으로 요소의 value 속성 변경
  - 이벤트 핸들링으로 받은 값을 데이터에 대입
  - file 타입에는 사용 불가

| v-model 장식자 | |
|---|---|
| 장식자 | 의미 |
| **.lazy** | input 대신 change 이벤트 핸들링 |
| **.number** | 값을 숫자로 변환 |
| **.trim** | 값 양쪽에 있는 쓸데없는 공백 제거 |

# 데이터 가공 및 감시

- 산출 속성; 리앤티브 데이터를 기반으로 결과 캐시
- getter and setter; get:, set:
- 워처(watcher); 특정 데이터 또는 산출 속성의 상태를 감시해서 변화가 있을 때 등록한 처리를 자동을 실행

```
new Vue({
  // ..
```

```
  watch: {
  <감시할 데이터>: function(<새로운 값>, <이전 값>) {
       // value가 변화했을 때 하고 싶은 처리
     },
     'item.value': function(newVal, oldVal) {
   // 객체의 속성도 감시할 수 있음
     },
     list: {
       handler: function(newVal, oldVal) {
         // list가 변경될 때 하고 싶은 처리
       },
       deep: true,  // 속성: deep, 값: Boolean, 네스트된 객체도 감시할지
       immediate: true  // 속성: immediate, 값: Boolean, 처음 값을 읽어 들이는 시점에
도 호출할지
     }
   }
})
```

- 인스턴스 메서드로 등록; this.$watch → 워처 제거
- 필터; 문자 수를 반올림하거나 쉼표를 넣는 등의 테스트 기반 변환 처리에 특화된 기능. v-bind 디렉티브 값 뒤에 파이프(|)로 연결해서 사용
    - 전역 필터; Vue.filter 사용해서 등록
- 사용자 정의 디렉티브; v- 프리픽스를 붙여 사용
    - directives:
    - 전역 등록; Vue.directive

| 사용자 정의 디렉티브에서 사용할 수 있는 훅 | |
|---|---|
| **메서드 이름** | **시점** |
| **bind** | 디렉티브가 처음 요소와 연결될 때 |
| **inserted** | 연결된 요소가 부모 Node에 삽입될 때 |
| **update** | 연결된 요소를 내포하고 있는 컴포넌트의 VNode가 변경되었을 때 |
| **componetUpdated** | 내포하고 있는 컴포넌트와 자식 컴포넌트의 VNode가 변경되었을 때 |
| **unbind** | 연결되어 있는 요소로부터 디렉티브가 제거될 때 |

| 훅 매개변수 | |
|---|---|
| **매개변수** | **내용** |
| **el** | 디렉티브가 연결되어 있는 요소 |
| **binding** | 배인드된 값, 매개변수, 장식자가 들어 있는 객체 |
| **vnode** | 요소에 대응되는 VNode |
| **oldVnode** | 변경 이전의 VNode(update 또는 componentUpdated에서만 사용 가능) |

| binding의 속성 | |
|---|---|
| **속성** | **설명** |
| **arg** | 매개변수 |
| **modifiers** | 장식자 객체 |
| **value** | 새로운 값 |
| **oldValue** | 이전 값(update 또는 componentUpdated에서만 사용 가능) |

- nextTick; Vue.nextTick, this.$nextTick

# Components

- Vue.component; 정의 → 사용
- 컴포넌트 간 통신;
  - 부모(속성, on) ↔ 자식( props, $emit)
  - 이벤트 버스
- 슬롯(slot); 부모가 되는 컴포넌트 측에서 자식이 되는 컴포넌트의 템플릿 일부를 끼워넣는 기능.
  - 이름 있는 슬롯
  - 스코프 있는 슬롯
- 컴포넌트의 v-model, .sync로 양방향 데이터 바인딩
- 템플릿; template 옵션, inline-template, text/x-template + 선택자, 단일 파일 컴포넌트, 렌더링 함수
- 템플릿이 DOM으로 인식되는 경우; 컴포넌트 명명 규칙, HTML 내포 가능한 요소 규칙
- 함수형 컴포넌트, 동적 컴포넌트, 공통적인 처리를 등록하는 믹스인

# Transition & Animation

- 트랜지션
  - 단일 요소 트랜지션

| 트랜지션 클래스의 시점 | |
|---|---|
| **Enter 계열의 클래스** | **대상 요소가 DOM에서 삽입될 때의 트랜지션 페이즈** |
| **.v-enter** | 대상 요소가 DOM에 삽입되기 전에 추가되며, 트랜지션이 종료될 때 사라짐. Enter 액티브 상태 |
| **.v-enter-to** | 트랜지션이 실제로 시작될 때 추가되며, 트랜지션이 종료될 때 사라짐. Enter의 종료 완료 |
| **.v-enter-active** | 대상 요소가 DOM에 삽입되기 전에 추가되며, 트랜지션이 종료될 때 사라짐. Enter의 액티브 상태 |
| **Leave 계열의 클래스** | **대상 요소가 DOM에서 제거될 때의 트랜지션 페이즈** |
| **.v-leave** | 트랜지션 시작 전에 추가되며, 트랜지션 개시 때는 사라짐. Leave 시작 상태 |
| **.v-leave-to** | 트랜지션 시작 전에 추가되어, 트랜지션이 종료되었을 때 사라짐. Leave 종료 상태 |
| **.v-leave-active** | 트랜지션 시작 전에 추가되어, 트랜지션이 종료되었을 때 사라짐. Leave의 액티브 상태 |

- 트랜지션 클래스의 기본
  - 추가할 때; .v-enter → .v-enter-to
  - 제거할 때; .v-leave → .v-leave-to

| Enter와 Leave 시점 변경 | |
|---|---|
| **모드** | **동작** |
| **in-out** | Enter 트랜지션이 종료되고 나서 Leave 트랜지션 시작 |
| **out-in** | Leave 트랜지션이 종료되고 나서 Enter 트랜지션 시작 |

- 리스트 트랜지션; 이동 트랜지션, Leave와 Move가 동시에 적용
- SVG 트랜지션
- 트랜지션 훅

| 사용할 수 있는 트랜지션 훅 | |
|---|---|
| **Enter 계열 훅** | **시점** |
| **before-enter** | DOM에 요소가 추가되기 전 |
| **enter** | .v-enter가 있는 DOM에 요소가 추가된 후 |
| **after-enter** | 트랜지션이 끝나거나 또는 enter에서 done()을 호출한 후 |
| **enter-cancelled** | enter 페이즈가 중간에 취소되었을 때 |
| **Leave 계열 훅** | **시점** |
| **before-leave** | 클래스가 추가되기 전 |
| **leave** | .v-leave가 추가된 후 |
| **after-leave** | DOM에서 요소가 제거된 후 또는 leave에서 done()을 호출한 후 |
| **leave-cancelled** | leave 페이즈가 중간에 취소되었을 때 |

# Applications

- Vuex; 상태 관리 전용 라이브러리, 여러 컴포넌트가 데이터를 공유할 수 있게 하여 애플리케이션 전체의 상태를 한 곳에서 관리할 수 있도록 함.
- Vue Router; 라우팅 전용 라이브러리, 컴포넌트로 구조화된 여러 개의 화면을 URL과 연결해서 SPA를 만들 수 있도록 함.
- Vue CLI; 애플리케이션 개발을 지원하기 위한 명령 라인 인터페이스
  - webpack 번들러; 모듈화한 여러 개의 파일을 모아 주는 번들러 https://webpack.js.org/
- SFC(Single File Components); html + javascript + css ⇒ .vue
- 다른 마크업 언어 또는 스타일시트 언어 사용; Pug, Sass 등 <alert type="info">npm install pug pug-loader –save-dev</alert>
- 공식 스타일 가이드; 파일, 사용자 정의 태그, 컴포넌트 이름 ⇒ 파스칼케이스(PascalCase)

## Node.js

- https://nodejs.org
- https://docs.npmjs.com/

**i**

npm install –save vue@2.6.14

| npm 기본 명령어 | | |
|---|---|---|
| **명령어** | **생략 기법** | **설명** |
| npm install –global | npm i -g | 전역 위치에 설치 |
| npm install –save | npm i -s | 프로젝트 결과물을 동작시키기 위해서 필요한 패키지를 설치 |
| npm install –save-dev | npm i -D | 개발 중에만 필요한 패키지 설치 |

- Babel; ECMAScript 표준과 JSX 트랜스파일러 https://babeljs.io/

## Vue CLI

- https://github.com/vuejs/vue-cli
- Vue CLI 템플릿; https://github.com/vuejs-templates

```
$ npm install -g vue-cli  # vue-cli 설치

$ vue --version # 버전 확인

# 프로젝트 생성 명령어의 기본 형태
$ vue init <템플릿 이름> <프로젝트 이름>
$ cd <프로젝트 이름>
$ npm install
```

```
$ vue init webpack my-app
```

- 프로젝트 루트; 폴더와 파일 구성
  - build 디렉토리; Webpack 빌드 전용 스크립트
  - config 디렉토리; Webpack 빌드 설정
  - dist 디렉토리; 빌드된 파일이 여기 들어감
  - **src 디렉토리**; 빌드할 파일을 넣음
    - **assets 디렉토리**; 이미지 또는 폰트를 넣음
    - **components 디렉토리**; 단일 파일 컴포넌트를 넣음
    - **router 디렉토리**; 라우팅과 관련된 설정
    - **App.vue 파일**; 애플리케이션 루트가 될 컴포넌트
    - **main.js 파일**; 엔트리 포인트
  - static 디렉토리; Loader를 사용하지 않고 곧바로 dist에 넣을 파일
  - index.html 파일; SPA의 인덱스가 되는 HTML 템플릿

```
$ npm run dev # 개발 서버 실행

$ npm run build # 프로젝트 빌드
```

### Vue.js 플러그인

```
// Vue와 Vuex 모듈 읽어 들이기
import Vue from 'vue'
import Vuex from 'vuex'

// Vue에 Vuex 등록하기
Vue.use(Vuex)
```

# Vuex

- Vuex; 상태 관리를 위한 확장 라이브러리
- 여러개의 컴포넌트가 데이터 공유
- 데이터 상태와 관련된 처리를 공통화

- 큰 상태 관리도 모듈을 사용해 간단하게 분리
- https://vuex.vuejs.org
- Vuex는 ES2015의 Promise를 사용하므로, 이를 지원하지 않는 브라우저에서도 지원할 수 있게 폴리필을 함께 설치; https://babeljs.io/docs/usage/polyfill

```
# 최신 버전 설치
$ npm install vuex babel-polyfill

# 버전 지정
$ npm install vuex@3.1.0 babel-polyfill@6.26.
```

```
import 'babel-polyfill'
import Vue from 'vue'
import Vuex from 'vuex'

// 플러그인으로 등록
Vue.use(Vuex)
```

- 스토어; Vuex 상태 관리를 위한 저장소

```
// 스토어 만들기
const store = new Vuex.store({
  state: {
    count: 0
  },
  mutations: {
  // 카운트 업하는 뮤테이션 등록, 매개변수로 전달
    increment(state) {
      state.count++
    }
    // increment: state => { state.count++ }  // 위와 같은 동작을 하는 코드
  }
})

export default store
```

```
import store from '@/store.js'  // @는 디폴트로 등록되어 있는 src 디렉토리의 별칭
console.log(store.state.count)  // -> 0

// 스토어의 상태 변경
// increment 커밋
store.commit('increment')
// 값이 변경된 것을 확인
console.log(store.state.count)  // -> 1
```

```
import store from './store.js'

new Vue({
  el: '#app',
  store,  // store 등록
  render: h => h(App)
})
```

```
export default {
  created() {
  // 스토어의 상태 확인
    console.log(this.$store.state.count)
  // 스토어의 상태 변경
    this.$store.commit('increment')
  }
}
```

- 컴포넌트에서 스테이트를 변경하고 싶으면 액션과 뮤테이션을 거쳐야 한다.
- 스테이트(state); 스토어에서 관리하고 있는 상태
- 게터(getter); 스테이트를 추출하기 위한 산출 데이터
- 뮤테이션; 컴포넌터의 methods.
    - state, payload 커밋에서 전달된 매개변수를 받음. 호출을 위해 타입과 핸들러 개념 사용.
    - 커밋(commit); 등록되어 있는 뮤테이션을 호출할 수 있게 해주는 인스턴스 메서드
- 액션(action); 비동기 처리를 포함할 수 있는 메서드. 데이터 가공 또는 비동기 처리를 실시한 후, 그 결과를 뮤테이션으로 커밋
    - 디스패치(dispatch); 등록되어 있는 액션을 호출하는 인스턴스 메서드
- Vuex의 규칙
    - 애플리케이션 레벨의 상태는 스토어로 관리
    - 상태를 변경하는 것은 뮤테이션 내부에서만 수행
    - 비동기 처리는 커밋하기 전에 완료
- 컴포넌트에서 스토어 사용
    - 메시지 사용
    - 메시지 변경
    - 상태와 게터에 v-model 사용
    - 컴포넌트와 스토어를 바인드하는 헬퍼
- 스토어 분할
    - 이름 공간
    - 헬퍼에 이름 공간 지정
    - 모듈 네스트
    - 이름 공간이 있는 모듈에서 외부에 접근
    - 모듈을 파일별로 분할
    - 모듈 재사용
- 이외의 기능과 옵션

- 스토어의 상태 감시
- strict 모드로 개발
- vuex에서 핫리로딩 사용

# Vue Router로 SPA 만들기

- Vue Router; 단일 페이지 어플리케이션 구축하기 위한 Vue.js 확장 라이브러리. 컴포넌트와 URL을 연결해 주는 기능.
- SPA(Single-page Application); 하나의 웹 페이지를 사용해서 요소의 내용만을 변경해 화면을 이동하는 애플리케이션 설계
- https://router.vuejs.org

Vue Router 설치

```
$ npm install vue-router # 최신 버전 설치

$ npm install vue-router@3.0.6 # 버전 지정
```

```
import Vue from 'vue'
import VueRouter from 'vue-router'

// 플러그인으로 등록
Vue.use(VueRouter)
```

| Vue Router 내장 컴포넌트 | |
|---|---|
| **사용자 정의 태그** | **설명** |
| <router-view> | 라우트와 일치하는 컴포넌트를 렌더링 |
| <router-link> | 라우트 링크를 생성 |

```
import Vue from 'vue'
import VueRouter from 'vue-router'

// 라우트 전용 컴포넌트 읽어 들이기
import Home from '@/views/Home.vue'
import Product from '@/views/Product.vue'

// Vuex와 마찬가지로 플러그인 등록
Vue.use(VueRouter)

// VueRouter 인스턴스 생성
const router = new VueRouter({
  // URL의 경로와 연결할 컴포넌트 맵핑하기
  routes: [
    { path: '/', component: Home },
    { path: '/product', component: Product }
```

```
  ]
})

// 생성한 VueRouter 인스턴스 익스포트
export default router
```

```
import router from './router.js'

new Vue({
  el: '#app',
  router,  // 애플리케이션 등록
  render: h => h(App)
})
```

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Home</router-link>
      <router-link to="/product">상품 정보</router-link>
    </nav>
  <!-- 여기에 경로에 일치하는 컴포넌트가 들어감 -->
    <router-view />
  </div>
</template>
```

- 해시(Hash), 히스토리(History) 두 가지 모드. 디폴트는 해시
- URL에 해시 붙이지 않기
- 아파치 서버; mod_rewrite 설정 추가
- 라우터 전용 속성; $router, $route
- 라우터 정의와 옵션
    - 이름 있는 라우터
    - 요청 매개 변수
    - 쿼리
    - 메타 필드
    - 리다이렉트
- 네비게이션
    - 템플릿으로 내비게이션
    - 프로그램으로 내비게이션

| 객체 형식으로 지정 | |
|---|---|
| **객체** | **설명** |
| name | 라우트 이름 |
| path | 라우트 경로 |
| params | 요청 매개변수 객체 |

| 객체 형식으로 지정 | |
|---|---|
| **객체** | **설명** |
| query | 쿼리 객체 |

| 액티브 링크 하이라이트 | |
|---|---|
| **클래스** | **설명** |
| .router-link-exact-active | 전체가 매치되는 라우트 |
| .router-link-active | 매치한 경로를 포함하는 라우트 |

| 프로그램으로 내비게이션 | |
|---|---|
| **메서드** | **설명** |
| push | 이력 엔트리 추가 |
| replace | 이력 엔트리 수정 |
| go | 브라우저 레벨에서 페이지 이동 |

- 매개 변수가 있는 동적 라우트
  - 패턴 매치 라우팅
  - 매개 변수를 props로 컴포넌트에 전달
  - 콘텐츠 출력; 상품 목록 출력, 상품정보 출력
- 네스트되어 있는 복잡한 페이지
  - 네스트된 라우트 정의
  - 데이터 공유에는 Vuex 사용
  - 부모 라우트 전용 컴포넌트 정의; 매개변수를 가진 링크를 만들 때는 이름 있는 라우트 사용, 네스트 때의 URL
- 내비게이션 가드
  - 내비게이션 가드의 매개변수; 이동하지 않을 때 next(false) 호출
  - 라우트 단위 가드
  - 전역가드
  - 컴포넌트 가드

| 내비게이션 가드의 매개변수 | |
|---|---|
| **매개 변수** | **설명** |
| to | 이동 후의 라우트 객체 |
| from | 이동 전의 라우트 객체 |
| next | 내비게이션 해결 전용 콜백 함수 |

| 라우트 단위 가드 | |
|---|---|
| **메서드 이름** | **시점** |
| beforeEnter | 라우트 이동 전 |

| 전역 가드 | |
|---|---|
| **메서드 이름** | **시점** |
| beforeEach | 모든 라우트의 이동 전, 컴포넌트 가드 해결 전 |
| beforeResolve | 모든 라우트의 이동 전, 컴포넌트 가드 해결 후 |
| afterEach | 모든 라우트의 이동 후 |

| 컴포넌트 가드 | |
|---|---|
| **메서드 이름** | **시점** |
| beforeRoutEnter | 해당 컴포넌트로 이동하기 전 |
| beforeRouteUpdate | 해당 컴포넌트에서 라우트가 변경되기 전 |
| beforeRouteLeave | 해당 컴포넌트에서 밖으로 이동하기 전 |

- 내비게이션 해결 흐름

1. 내비게이션이 트리거됨
2. 비액티브화된 컴포넌트에서 leave 가드를 호출
3. 전역 beforeEach 가드를 호출
4. 재사용되는 컴포넌트에서 beforeRouteUpdate 가드를 호출
5. 라우트 설정 내부의 beforeEnter를 호출
6. 비동기 라우트 컴포넌트를 해결
7. 액티브화된 컴포넌트에서 beforeRouteUpdate를 호출
8. 전역 beforeResolve 가드를 호출
9. 내비게이션이 확정
10. 전역 afterEach 훅을 호출
11. DOM 변경이 트리거 됨
12. beforeRouteEnter의 next로 전달된 콜백 호출

- 페이지 이동 효과 적용
  - 간단한 트랜지션
  - 비동기 처리를 포함한 트랜지션
- 자주 사용하는 기능과 옵션
  - 이동 전에 컴포넌트 읽어 들이기
  - 비동기로 컴포넌트 읽어 들이기
  - 라우트 접근 제한
  - 스크롤 동작 조작

# 컴포넌트 데이터 전달

```
<template>
  <div>
    <ChildComponent />
  </div>
</template>

<script>
import ChildComponent from "@/components/childComponent.vue";

export default {
  name: "ParentComponent",
  components: { ChildComponent },
};
</script>
```

- [Vue.js 시작하기] - 컴포넌트 데이터 전달 방법

# References

- SSR를 사용한 고속 초기 렌더링, OGP 대응
- PWA를 사용한 애플리케이션 형태의 스마트폰 사이트
- Electron을 사용한 데스크톱 애플리케이션
- NativeScript를 사용한 스마트폰 애플리케이션

- 고양이도 할 수 있는 Vue.js 지원 페이지
- 基礎から学ぶ Vue.js 書籍用サポートページ 基礎から学ぶ Vue.js 書籍用サポートページ
- Vue.js 공식문서
- Vuex 공식문서
- Vue Router 공식문서
- Vue.js Ver2 스타일 가이드
- MDN 웹문서

1)

kebab-case, lisp-case, spinal-case; 하이픈으로 구분

From:
https://theta5912.net/ - **reth**

Permanent link:
**https://theta5912.net/doku.php?id=public:computer:vuejs&rev=1638860469**

Last update: **2021/12/07 16:01**