

Vue.js 데이터 호출 패턴

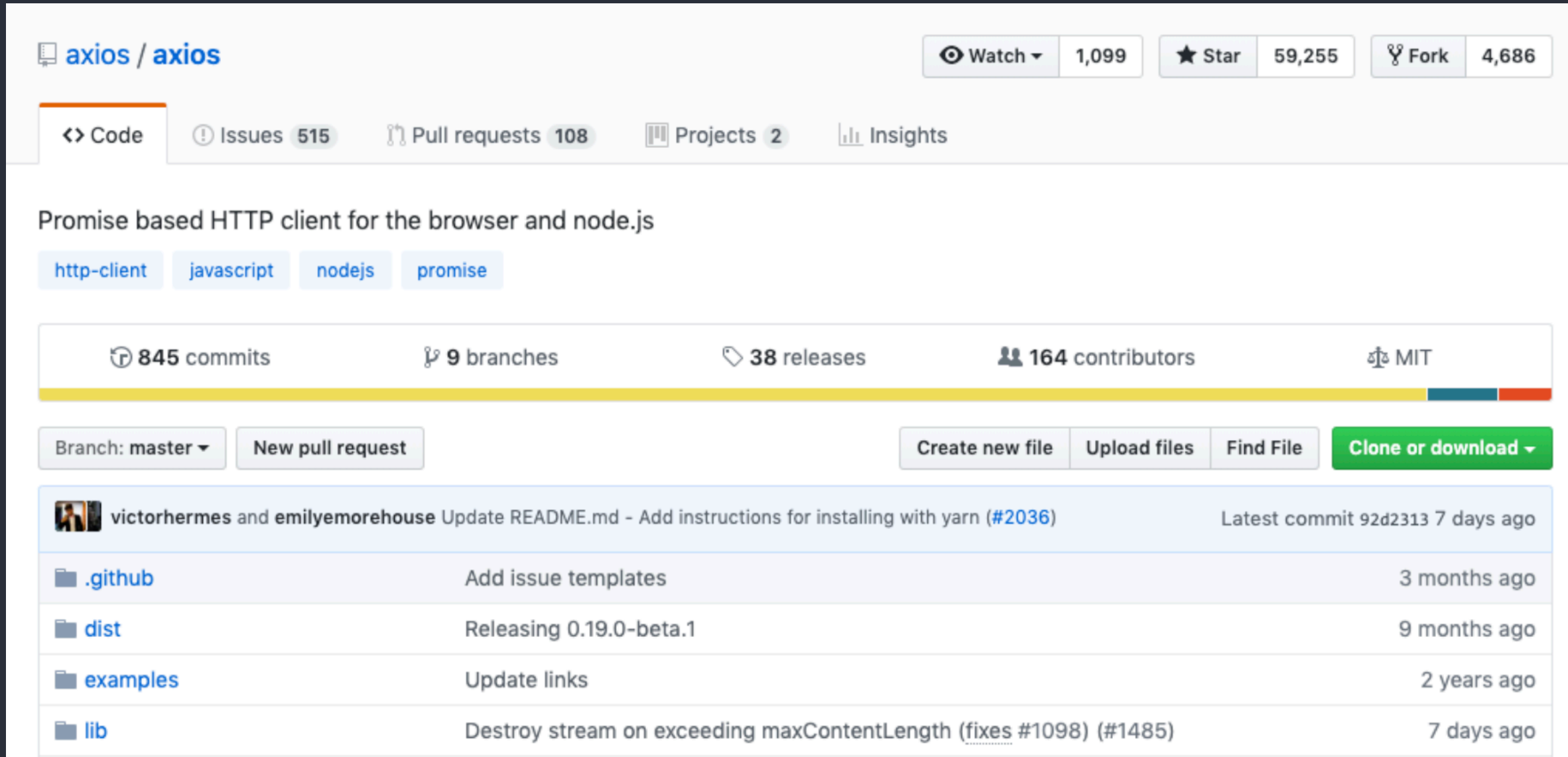
목차

- ▶ 데이터 호출 라이브러리 axios
- ▶ 데이터 호출 패턴 I - Instance Lifecycle
 - ▶ 인스턴스 라이프 사이클
- ▶ 데이터 호출 패턴 II - Navigation Guard
 - ▶ 뷰 라우터
 - ▶ 뷰엑스

데이터 호출 라이브러리 axios

axios

뷰에서 가장 많이 사용되는 HTTP 통신 라이브러리



The screenshot shows the GitHub repository for axios. At the top, the repository name 'axios / axios' is displayed. To the right, there are buttons for 'Watch' (1,099), 'Star' (59,255), and 'Fork' (4,686). Below this, a navigation bar includes 'Code', 'Issues' (515), 'Pull requests' (108), 'Projects' (2), and 'Insights'. The repository description is 'Promise based HTTP client for the browser and node.js', with tags for 'http-client', 'javascript', 'nodejs', and 'promise'. A summary bar shows '845 commits', '9 branches', '38 releases', '164 contributors', and the 'MIT' license. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find File', and 'Clone or download'. The commit history section shows a recent commit by 'victorhermes and emilyemorehouse' updating the README.md with instructions for installing with yarn. Below the commit history, there is a table of recent releases.

File	Commit Message	Time Ago
.github	Add issue templates	3 months ago
dist	Releasing 0.19.0-beta.1	9 months ago
examples	Update links	2 years ago
lib	Destroy stream on exceeding maxContentLength (fixes #1098) (#1485)	7 days ago

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원
- ▶ **Promise** 기반의 API 제공

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원
- ▶ **Promise** 기반의 API 제공
- ▶ **Instance** 기반의 **설정 옵션 확장**

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원
- ▶ **Promise** 기반의 API 제공
- ▶ **Instance** 기반의 **설정 옵션 확장**
- ▶ **Inteceptor**를 이용한 Request, Response **전처리**

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원
- ▶ **Promise** 기반의 API 제공
- ▶ **Instance** 기반의 **설정 옵션 확장**
- ▶ **Inteceptor**를 이용한 Request, Response **전처리**
- ▶ 우수한 사용 예시 및 **문서화**

axios 특징

- ▶ CDN, NPM 설치 방식 모두 지원
- ▶ **Promise** 기반의 API 제공
- ▶ **Instance** 기반의 **설정 옵션 확장**
- ▶ **Inteceptor**를 이용한 Request, Response **전처리**
- ▶ 우수한 사용 예시 및 **문서화**
- ▶ **다양한 케이스에 대한 API와 옵션 지원**

axios 특징 - Promise 기반 API

axios

```
.get('http://domain.com/users/')  
.then(function(response) {  
    // 호출한 데이터 처리  
})  
.catch(function(error) {  
    // 호출시 발생한 에러 처리  
});
```

axios 특징 - Instance 기반 옵션 확장

```
const instance = axios.create({  
  baseURL: 'https://domain.com/',  
  timeout: 5000,  
  headers: { 'X-Custom-Header': 'temp' },  
  // ...  
});
```

axios 특징 - Interceptor를 이용한 전처리

```
instance.interceptors.request.use(  
  // 서버로 요청이 가기 전의 처리  
  function(config) {},  
  // 요청이 실패 했을 때의 처리  
  function(error) {},  
);
```

```
instance.interceptors.response.use(  
  // 응답 데이터가 컴포넌트에 도달하기 전의 처리  
  function(response) {},  
  // 데이터가 정상적으로 수신되지 않았을 때의 처리  
  function(error) {},  
);
```

axios 특징 - Interceptor를 이용한 전처리

```
instance.interceptors.request.use(  
  // 서버로 요청이 가기 전의 처리  
  function(config) {  
    var token = getToken();  
    config.headers.Authorization = token;  
    return config;  
  },  
  // 요청이 실패 했을 때의 처리  
  function(error) {  
    handleError(error);  
    return Promise.reject(error.response);  
  },  
);
```

axios 특징 - 우수한 문서화와 사용 예시

Request Config

These are the available config options for making requests. Only the `url` is required. Requests will default to `GET` if `method` is not specified.

```
{
  // `url` is the server URL that will be used for the request
  url: '/user',

  // `method` is the request method to be used when making the request
  method: 'get', // default

  // `baseUrl` will be prepended to `url` unless `url` is absolute.
  // It can be convenient to set `baseUrl` for an instance of axios to pass relative URLs
  // to methods of that instance.
  baseUrl: 'https://some-domain.com/api/',

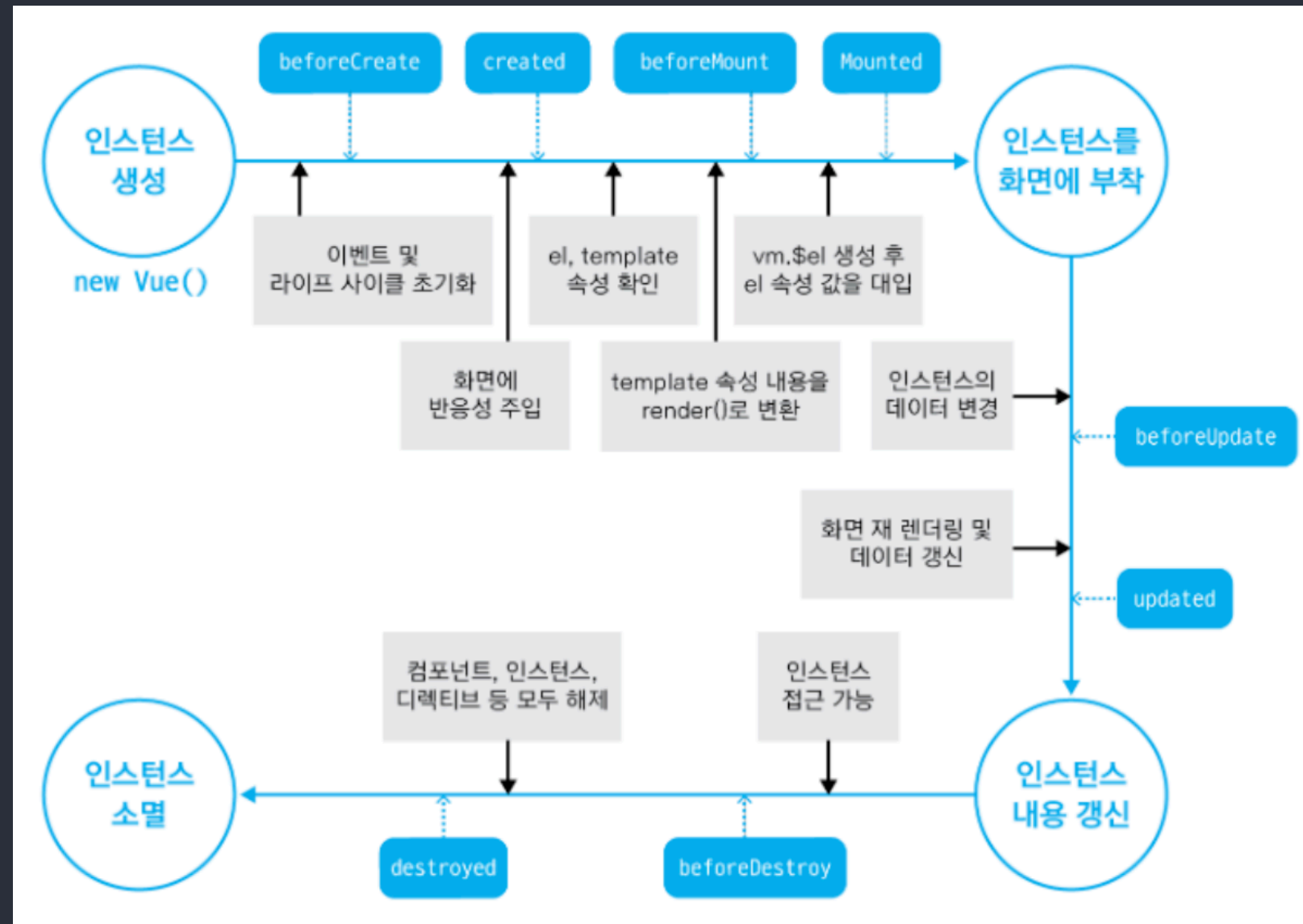
  // `transformRequest` allows changes to the request data before it is sent to the server
  // This is only applicable for request methods 'PUT', 'POST', and 'PATCH'
  // The last function in the array must return a string or an instance of Buffer, ArrayBuffer,
  // FormData or Stream
  // You may modify the headers object.
```

데이터 호출 패턴 I

Instance Lifecycle

뷰 인스턴스 라이프 사이클

뷰 인스턴스가 생성되고 소멸되기까지의 생애 주기



라이프 사이클 훅

라이프 사이클 각 단계에서 실행되는 함수 또는 커스텀 로직

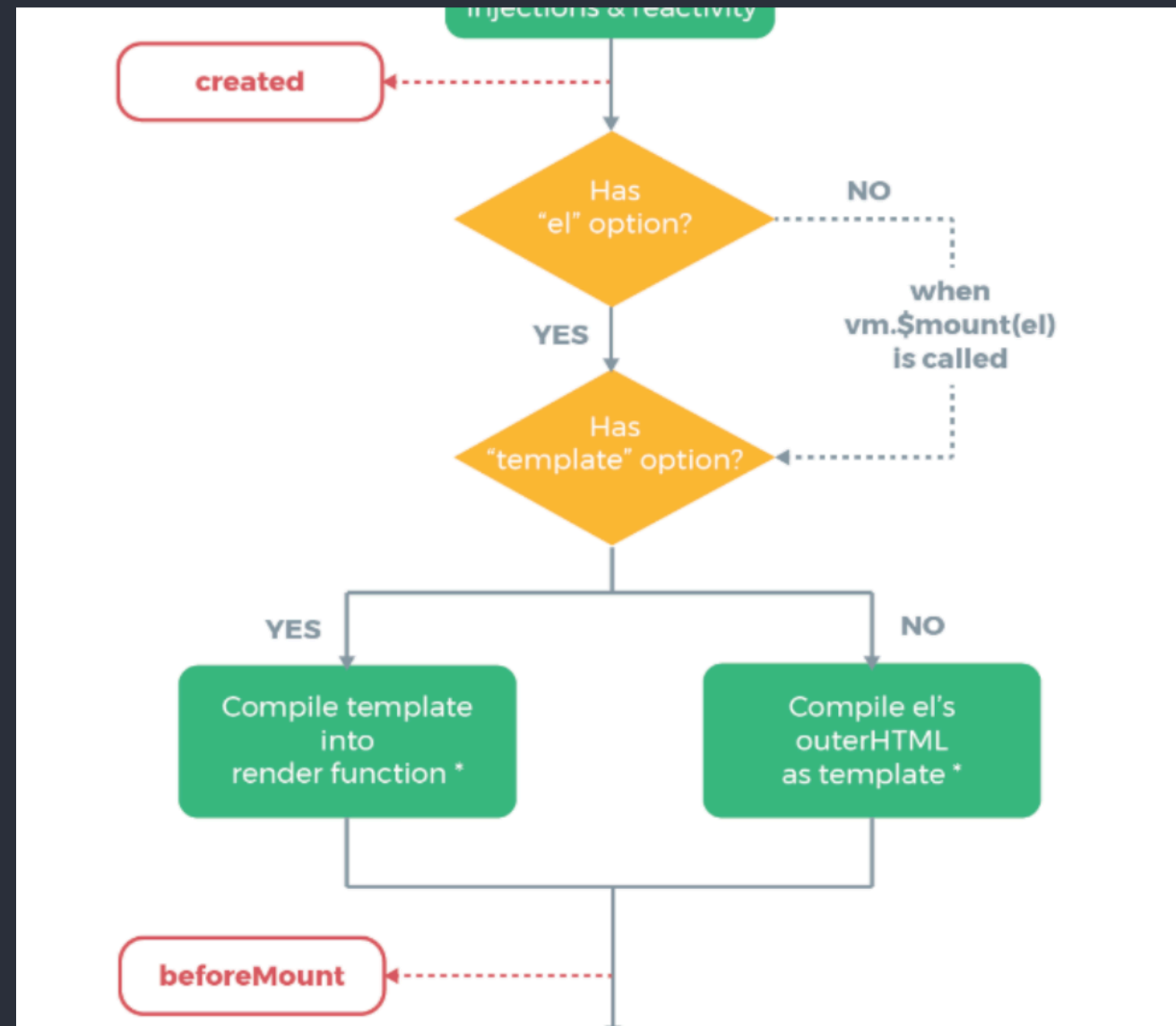
```
new Vue({  
  created: function() {  
    console.log('생성 되었습니다');  
  },  
  beforeDestroy: function() {  
    console.log('이제 파괴됩니다');  
  },  
});
```

라이프 사이클 훅을 이용한 데이터 호출

```
new Vue({  
  methods: {  
    fetchData: function() {  
      axios.get('domain.com/users/1');  
    },  
  },  
  created: function() {  
    this.fetchData();  
  },  
});
```

데이터 호출에 적합한 라이프 사이클 훅

created, beforeMount



데이터 호출 패턴 II

Router Navigation Guard

뷰 라우터

뷰에서 **페이지 이동**과 관련된 **API 및 속성**을 제공하는 공식 라이브러리

```
new VueRouter({  
  mode: 'history',  
  routes: [  
    {  
      path: '/products',  
      component: ProductList,  
    },  
  ],  
});
```

뷰 라우터 네비게이션 가드

특정 페이지로 **진입하기 전** **커스텀 로직**을 반영할 수 있는 속성

```
new VueRouter({
  routes: [
    {
      path: '/products',
      component: ProductList,
      beforeEnter: function(to, from, next) {
        if (!token) {
          next('/login');
          return;
        }
        next();
      },
    },
  ],
});
```

뷰 라우터 네비게이션 가드

특정 페이지로 **진입하기 전** **커스텀 로직**을 반영할 수 있는 속성

```
new VueRouter({
  routes: [
    {
      path: '/products',
      component: ProductList,
      beforeEnter: function(to, from, next) {
        if (!token) {
          next('/login');
          return;
        }
        next();
      },
    },
  ],
});
```


스토어와 라우터 네비게이션 가드를 이용한 데이터 호출

```
new VueRouter({
  routes: [
    {
      path: '/products',
      component: ProductList,
      beforeEnter: function(to, from, next) {
        store
          .dispatch('FETCH_PRODUCTS')
          .then(function() {
            next();
          })
          .catch(function() {
            alert('fetching data failed');
          });
      },
    },
  ],
});
```

스토어와 라우터 네비게이션 가드를 이용한 데이터 호출

```
new VueRouter({
  routes: [
    {
      path: '/products',
      component: ProductList,
      beforeEnter: function(to, from, next) {
        store
          .dispatch('FETCH_PRODUCTS')
          .then(function() {
            next();
          })
          .catch(function() {
            alert('fetching data failed');
          });
      },
    },
  ],
});
```

