**Cache dữ liệu lấy từ API qua ví dụ đơn giản**

5.667 lượt xem - Thời gian đọc: 4 phút đọc

Cache không còn là một từ quá xa lạ đối với một lập trình viên web. Chúng ta hay nghe tới nó khi xử lý các yêu cầu truy vấn ở phía máy chủ khi xử lý các yêu cầu có tính thường xuyên và lặp dữ liệu. Đối với việc xử lý yêu cầu lấy dữ liệu từ phía máy chủ, đôi khi cũng trả về các phản hồi dữ liệu lặp lại, tốn kha khá thời gian nhận yêu cầu ở phía Front-end. Để tối ưu điều đó, đôi khi chúng ta cũng cần cache lại các network request ở phía Front-end, giúp tối ưu yêu cầu truy xuất dữ liệu ở cả 2 phía.

Một giải pháp tương đối đơn giản mà bạn có thể sử dụng với **Axios** . Ý tưởng vô cùng đơn giản, có một bộ nhớ cache, có thể lưu trữ các đối tượng thuộc bất kỳ loại nào. Sau đó, làm mất hiệu lực của chúng nếu chúng đã được lưu trữ trong một khoảng thời gian dài hơn thời gian lưu vào bộ nhớ đệm.

**Lưu ý:** Bài viết dừng lãi mức tham khảo, khuyến khích các bạn nên tích hợp các Caching Network Package vì nó ổn định và dễ bảo trì về sau cho mã nguồn.

**Tạo bộ xử lý cache sử dụng localStorage**

Tại Front-end, chúng ta có thể sử dụng LocalStorage như một bộ nhớ tạm. Bây giờ, chúng ta hãy cùng tạo 1 tệp **src/core/modules/cacheStore.js**

class CachStore {}

const cacheStore = new CachStore();

export default cacheStore;

Định nghĩa một thức bên trong đối tượng CacheStore **remember** để lưu giá trị được **fetch** từ API về và thời gian hết hạn của dữ liệu cache

remember(key: string, value: any) {

const timeExpired = Date.now() + 1000 \* 60 \* 1;

const \_value = `${JSON.stringify(value)}((@))${timeExpired}`;

localStorage.setItem(key, \_value);

}

Sau khi định nghĩa một phương thức để lưu dữ liệu vào bộ nhớ tạm của chúng ta, sẽ cần thức để lấy dữ liệu được cache ra. Chúng ta hãy định nghĩa thêm một hàm **get**

get(key: string) {

// Lấy dữ liệu từ store

const value = localStorage.getItem(key);

if (!value) {

return null;

}

const splitting = value.split("((@))");

const expired: any = splitting[1]; // Lấy giá trị

const cacheValue: string = splitting[0]; // Lấy thời gian hết hạn

// Kiểm tra thời gian đính kèm nếu quá hạn, sẽ trả về null

// Khi gửi request, chúng ta sẽ kiểm tra nếu giá trị get null, chúng ta sẽ lấy dữ liệu api và set lại cho cacheStore

if (expired <= Date.now()) {

localStorage.removeItem(key);

return null;

} else {

// Trả về dữ liệu đã cache nếu tồn tại

return cacheValue ? JSON.parse(cacheValue) : null;

}

}

Khi gôm chung lại, chúng ta sẽ được tổng thể class CacheStore của chúng ta như bên dưới

class CacheStore {

// thêm dữ liệu vào cache

remember(key: string, value: any) {

const timeExpired = Date.now() + 1000 \* 60 \* 1;

const \_value = `${JSON.stringify(value)}((@))${timeExpired}`;

localStorage.setItem(key, \_value);

}

// Lấy dữ liệu từ cache

get(key: string) {

const value = localStorage.getItem(key);

if (!value) {

return null;

}

const splitting = value.split("((@))");

const expired: any = splitting[1];

const cacheValue: string = splitting[0];

if (expired <= Date.now()) {

localStorage.removeItem(key);

return null;

} else {

return cacheValue ? JSON.parse(cacheValue) : null;

}

}

pull(key: string) {

const value = this.get(key);

localStorage.removeItem(key);

return value;

}

}

const cacheStore = new CacheStore();

export default cacheStore;

**Định nghĩa đối tượng gọi API**

Để hạn chế việc hao tốn tài nguyên, ở đây mình sẽ sử dụng MockAPI để demo cho bài viết này.

https://5ebbd7c6f2cfeb001697d2bf.mockapi.io/api/v1/articles

Bây giờ mình sẽ định nghĩa một đối tượng **MockService** có nhiệm vụ gọi lên **API** và lấy dữ liệu thông qua phương thức **getList**

import axios, { AxiosResponse } from "axios";

export interface Article {

id: string;

createdAt: string;

title: string;

description: string;

paragraph: string;

thumbnail: string;

author: string;

}

class MockService {

async getList() {

try {

const response: AxiosResponse<Article> = await axios.get(

"https://5ebbd7c6f2cfeb001697d2bf.mockapi.io/api/v1/articles"

);

return response.data;

} catch (e) {

return Promise.reject(e);

}

}

}

Tại đối tượng này, mình sẽ import **storeCache** mà chúng ta đã định nghĩa và thêm các trường hợp để cache và cũng như lấy dữ liệu đã được cache

* **Trường hợp 1:** cache trống: Sẽ lấy dữ liệu từ API sau đó thêm vào storeCache bằng phương thức **remember**
* **Trường hợp 2:** cache đã có: Sẽ lấy dữ liệu từ cache, nếu chưa hết hạn
* **Trường hợp 3:** lấy dữ liệu từ cache nhưng đã hết hạn, sẽ lấy dữ liệu từ API lại và thực hiện lại trường hợp đầu tiên

class MockService {

async getList() {

try {

// Trường hợp 2 và 3

if (cacheStore.get("mock-service-get")) {

console.log("run cache");

return cacheStore.get("mock-service-get");

}

// Trường hợp 1

console.log("fetching");

const response: AxiosResponse<Article> = await axios.get(

"https://5ebbd7c6f2cfeb001697d2bf.mockapi.io/api/v1/articles"

);

// Trường hợp 1

cacheStore.remember("mock-service-get", response.data);

return response.data;

} catch (e) {

return Promise.reject(e);

}

}

}

Sau khi xây dựng bước này, chúng ta đã hoàn thành 90% tính năng Cache network request. Để thấy kết quả rõ nhất. Mình sẽ đặt link demo về case này ở đây. Chúng ta sẽ kết hợp phương pháp trên với ReactJS nhé. [Demo](https://codesandbox.io/s/cache-request-3soft?file=/src/App.tsx&fbclid=IwAR0n8WJtW3QcZKZi2xwcWzSxExg3YmKyLw22JEGpX4eyzgdolXWILtrOLH4)