

Хуки жизненного цикла экземпляра компонента. Composable-функции. Запросы и пагинация.

Вы уже изучили, что такое состояние компонента, реактивность этого состояния, а также способы переиспользования компонентов. Однако во Vue 3 Вы также можете переиспользовать и логику. Для этого создаются специальные функции, инкапсулирующие некоторую логику и состояния и возвращающие эти состояния с сохранением реактивности. Такие функции называются composable-функциями. Их принято именовать с префиксом «use», например, «useClipboard».

Ниже представлен код composable-функции «useClipboard»:

```
clipboard.js 948 B

1  import { NOTICE_TYPES, useNotice } from "../notice";
2
3  export const useClipboard = () => {
4    const { throwNotice } = useNotice();
5
6    const getFileFromClipboard = async ({ fileName = "", typeRegEx = "" } = {}) => {
7      try {
8        const items = await navigator.clipboard.read();
9        const blob = await items[0].getType(items[0].types[0]);
10
11        const isValid = !typeRegEx || typeRegEx.test(blob.type);
12
13        if (!isValid) {
14          throw new Error("Incorrect file MIME-type");
15        }
16
17        const name = fileName || `clipboard.${blob.type.split("/")[1]}`;
18        const file = new File([blob], name, { type: blob.type });
19
20        return file;
21      } catch (err) {
22        throwNotice({
23          type: NOTICE_TYPES.ERROR,
24          text: err,
25        });
26
27        return Promise.reject(err);
28      }
29    };
30
31    return { getFileFromClipboard };
32  };
33
```

Composable-функция может импортировать и использовать другие composable-функции, что позволяет более эффективно переиспользовать код.

Также необходимо отметить, что composable-функция может принимать аргументы, являющиеся ref-ссылками или функциями-геттерами. Для этого существует метод `toValue()`, принимающий на вход либо простую переменную, либо `ref`, либо функцию. В случае `ref` он возвращает его свойство `value`, в случае функции он возвращает возвращенное этой функцией значение. В остальных случаях метод `toValue()` возвращает само переданное ему значение.

В документации [1] хорошо разобраны примеры использования composable-функций «`useMouse`» и «`useFetch`». Внимательно изучите их, а также обратите внимание на раздел, посвященный разбору ограничений в использовании таких функций.

Ещё один важный функционал, который предоставляет фреймворк Vue – это перехват событий жизненного цикла экземпляра компонента, таких как инициализация реактивного состояния, монтирование в DOM, размонтирование и другие. Такой перехват осуществляется путём передачи коллбэков в специальные функции, называемые хуками (Hooks). Например, если экземпляр компонента монтируется в DOM-дерево, то будет вызван коллбэк, переданный в функцию `onMounted()`. В документации [2] для наглядности изображена диаграмма последовательности срабатывания хуков. В [3] описано более подробно о каждом хуке.

Вопросы и задания

1. Для реализации основного функционала приложения `notes-app` необходимо реализовать отправку запросов для получения данных с сервера. Сейчас Вы развернёте простой HTTP-сервер у себя на компьютере. Для этого клонируйте репозиторий [4] с помощью команды «`git clone`», перейдите в директорию с сервером и следуйте инструкциям в разделах «`Setup`» и «`Launch`», приведённых в файле `README.md`. В результате в консоли должно появиться сообщение вида «`Server running at http:// 127.0.0.1:5000/`», что означает успешность запуска сервера. Теперь вы можете отправлять запросы на эндпоинты (Endpoints) из приложения `notes-app`. Список доступных эндпоинтов представлен в файле `README.md`.

2. Создайте поддиректорию `/composables` в директории `/src` проекта `notes-app`. Создайте в этой поддиректории файл `request.js`, экспортирующий `composable`-функцию `useRequest`. Функция `useRequest` должна принимать аргументы:

- `url`;
- `method` – метод запроса;
- `headers` – заголовки запроса;
- `query` – объект или список `query`-параметров запроса;
- `body` – тело запроса

и возвращать:

- `request` – асинхронная функция запроса;
- `data` – полученные данные в случае успешного запроса;
- `error` – текст ошибки в случае неуспешного запроса;
- `isInit` – реактивное булево значение «запрос еще не начат»;
- `isLoading` – реактивное булево значение «запрос начат, но не завершён»;
- `isLoaded` – реактивное булево значение «запрос завершён успешно и данные получены или отправлены»;
- `isError` – реактивное булево значение «запрос завершён неуспешно и возвращена ошибка».

Похожее задание вы уже выполняли в теме «Запросы к серверу. XHR и Fetch API» (задание №6). Текущая функция должна работать аналогично.

3. Воспользуйтесь написанной в предыдущем задании `composable`-функцией «`useRequest`» в компоненте `/src/views/notes.vue`, чтобы выполнить запрос на получение списка заметок. Соответствующий эндпоинт указан в файле `README.md`. Самостоятельно выберите точку вызова данной функции и обоснуйте свой выбор.

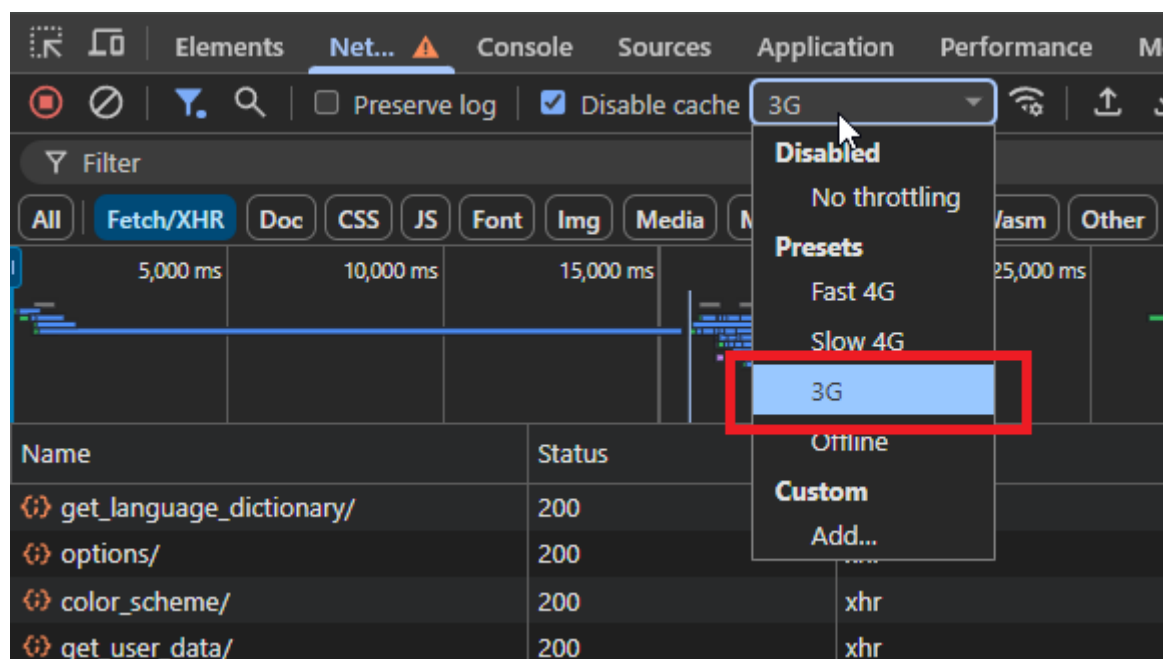
Удалите статические данные из `ref`-ссылки на список заметок и присвойте ей массив из запроса. Не забудьте обработать состояние для вывода общего кол-ва заметок в футере.

4. Реализуйте пагинацию списка заметок через `query`-параметры запроса. Пагинация должна быть непрерывной – при скролле страницы вниз должен подгружаться следующий массив

заметок. Добавьте ref-ссылку на отслеживание текущего номера страницы и ref-ссылку для хранения значения, введённого в строку поиска. Также вам понадобится ознакомиться с Intersection Observer API в документации [5] для реализации непрерывной пагинации. При вводе текста в строку поиска страница пагинации должна сбрасываться до единицы, а список заметок очищаться.

Создайте SFC-компонент прелоадера в директории /src/components. Стилизацию и вид прелоадера выберите на свое усмотрение. Данный компонент должен отображать прелоадер в конце списка каждый раз, когда идёт загрузка следующего массива заметок в процессе пагинации. Здесь Вам понадобятся статусы запроса, возвращаемые функцией useRequest.

В качестве референса можете заглянуть на платформу «Вовлекай» [6] в раздел «База знаний». Чтобы увидеть прелоадер в конце списка Вам может понадобиться принудительно «замедлить» сеть, чтобы увидеть прелоадер. Это можно сделать во всех браузерах через инструменты разработчика:



Например, в Google Chrome во вкладке «Network» (см. скриншот выше).

Ссылки

- [1]. <https://ru.vuejs.org/guide/reusability/composables.html>
- [2]. <https://ru.vuejs.org/guide/essentials/lifecycle.html>
- [3]. <https://ru.vuejs.org/api/composition-api-lifecycle.html>
- [4]. <https://github.com/GreysMouse/notes-app-server.git> (HTTP)
<git@github.com:GreysMouse/notes-app-server.git> (SSH)
- [5]. https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API
- [6]. <https://labmedia.vovlekay.online/game/library/>