
vuex-cache

Cache dispatched actions and prevent repeated requests and heavy actions.

Compatibility

- `Map` and `Promise` are required (you can use polyfills, like `@babel/polyfill`);
- Any Vue version, since `vuex-cache` just deals with Vuex;
- Vuex versions 1, 2 and 3.

Installation

`vuex-cache` is published in the NPM registry and can be installed using any compatible package manager.

```
1 npm install vuex-cache --save
2
3 # For Yarn use the command below.
4 yarn add vuex-cache
```

Import `createCache` factory and use on Vuex's plugins.

```
1 import Vue from 'vue';
2 import Vuex, { Store } from 'vuex';
3 import createCache from 'vuex-cache';
4
5 const store = new Store({
6   plugins: [createCache()],
7   ...
8 });
```

Installation on Nuxt.js

Global plugin (~plugins/vuex-cache.js)

Only use it if you're not using Classic Mode.

Create a module on plugins to setup `vuex-cache`. Call `vuex-cache` with your options, then call returned value with store on `onNuxtReady` event.

~/plugins/vuex-cache.js

```
1 import createVuexCache from 'vuex-cache';
2
3 export default ({ store }) => {
4   const options = {
5     timeout: 2 * 60 * 60 * 1000 // Equal to 2 hours in milliseconds.
6   };
7
8   const setupVuexCache = createVuexCache(options);
9
10  window.onNuxtReady(() => setupVuexCache(store));
11 };
```

Then just add this plugin to your nuxt configuration. Like the example below.

~/nuxt.config.js

```
1 module.exports = {
2   ...,
3   plugins: [
4     ...,
5     { src: '~/plugins/vuex-cache.js', ssr: false },
6   ]
7 };
```

Global store (~store/index.js)

```
1 import createCache from 'vuex-cache';
2
3 export const plugins = [
4   createCache()
5 ]
```

Usage

After install you can use `cache` property to call cache methods.

```
1 const store = new Store({
2   ...,
3   actions: {
4     'FETCH_USER': async (_, id) => {
5       const response = await fetch(baseUrl + '/user/' + id);
6       const { users } = await response.json();
7       return users;
8     }
9   }
10 });
11
12 store.cache.dispatch('FETCH_USER', 1);
13 //=> Promise { User }
```

API

createCache

The default exported factory to create [Vuex](#)'s store plugin. It define [cache](#) property on Store instances.

```
1 import { Store } from 'vuex';
2 import createCache from 'vuex-cache';
3
4 const store = new Store({
5   plugins: [
6     createCache()
7   ]
8 })
```

cacheAction

A named exported function to enhance actions and define [cache](#) property on ActionContext instances.

```
1 import { cacheAction } from 'vuex-cache';
2
3 // ...
4
5 const actions = {
6   'FETCH_STARGAZERS': cacheAction(
7     ({ cache, commit }, payload) => (
8       cache.dispatch('FETCH_REPOSITORIES')
9         .then((repos) => Promise.all(repos.map(getStargazers)))
10        .then((stargazers) => {
11          commit('STARGAZERS', [].concat(...stargazers));
12        })
13     )
14   ),
15
16   'SET_STARGAZERS': (context, payload) => { ... }
17 }
```

store.cache.dispatch

Dispatches an action if it's not cached and set it on cache, otherwise it returns cached [Promise](#).

It uses action **name** and **payload** as cache key.

```
1 store.cache.dispatch('user/GET_USER');
2 //=> Promise { User }
3
4 // Returns value without dispatching the action again.
5 store.cache.dispatch('user/GET_USER');
6 //=> Promise { User }
```

store.cache.has

Check if an action is cached. Returns **true** if action is cached and **false** otherwise.

```
1 store.cache.has('user/GET_USER');
2 //=> true
3
4 store.cache.has('FETCH_REPOSITORY', 219);
5 //=> false
```

store.cache.delete

Delete an action from cache. Returns **true** if action is deleted and **false** otherwise.

```
1 store.cache.delete('user/GET_USER');
2 //=> true
3
4 store.cache.delete('FETCH_REPOSITORY', 219);
5 //=> false
```

Only exact matches are deleted. Use `store.cache.clear` to delete all items or by action name.

store.cache.clear

Clear the cache, delete all actions from it. Returns **true** if cache is cleared and **false** otherwise.

```
1 store.cache.clear();
2 //=> true
```

If using the type parameter, only actions with the specified type are deleted from cache and the number of deleted keys is returned.

```
1 // store.cache.dispatch('FETCH_REPOSITORIES', { page: 1 });
2 // store.cache.dispatch('FETCH_REPOSITORIES', { page: 2 });
3 store.cache.clear('FETCH_REPOSITORIES');
4 //=> 2
```

store.cache.state

Warning! Don't use this method in production.

Helper method for debugging. Prints the current value of the cache (state).

```
1 store.cache.dispatch('FETCH_REPOSITORIES', { page: 1 });
2 store.cache.dispatch('FETCH_REPOSITORIES', { page: 2 });
3 store.cache.state();
4 //=> Map(2){...}
```

mapCacheActions

Create component methods that dispatch a cached action.

```
1 import { mapCacheActions } from 'vuex-cache';
2
3 export default {
4   name: 'Users',
5   methods: {
6     ...mapCacheActions(['FETCH_REPOSITORY']),
7     ...mapCacheActions('user', ['GET_USER']),
8   },
9   async mounted() {
10     this.GET_USER();
11     this.FETCH_REPOSITORY(219, {
12       timeout: 30000
13     });
14   }
15 }
```

Payload

The payload value is `undefined` as default and supports functions, primitive values and JSON parseable objects.

`store.cache.dispatch`, `store.cache.has` and `store.cache.delete` supports payload object as argument.

```
1 store.cache.dispatch({
2   type: 'FETCH_REPOSITORY',
3   payload: 198
4 });
5 //=> Promise { Repository }
6
7 store.cache.has({
8   type: 'FETCH_REPOSITORY',
9   payload: 198
10 });
11 //=> true
12
13 store.cache.delete({
14   type: 'FETCH_REPOSITORY',
15   payload: 198
16 });
17 //=> true
```

Timeout

`timeout` option is 0 as default and define cache duration is milliseconds.

0 means it has no defined duration, no timeout.

```
1 const store = new Store({
2   plugins: [
3     createCache({ timeout: 10000 })
4   ],
5   ...
6 });
```

After milliseconds defined in timeout option an action is expired from cache.

```
1 // This dispatches the action and set it on cache.
2 store.cache.dispatch('FETCH_REPOSITORY', 219);
3 //=> Promise { Repository }
4
5 store.cache.has('FETCH_REPOSITORY', 219);
6 //=> true
7
8 setTimeout(() => {
9
10   // It returns false because the action is expired.
11   store.cache.has('FETCH_REPOSITORY', 219);
12   //=> false
13
14   // This dispatches the action again because the action is expired.
15   store.cache.dispatch('FETCH_REPOSITORY', 219);
```

```
16 //=> Promise { Repository }  
17 }, 10000)
```

Store's timeout can be overwritten by dispatch timeout option in Dispatch Options or in payload.

```
1 store.cache.dispatch('FETCH_REPOSITORY', 219, {  
2   timeout: 30000  
3 });  
4  
5 // OR  
6  
7 store.cache.dispatch({  
8   type: 'FETCH_REPOSITORY',  
9   payload: 219,  
10  timeout: 30000  
11 });
```