Pinia Store Reference Guide

Core Store Properties

Property/Method	Description	Example Usage
`$state`	Access or replace the whole state of the store	`store.$state = { count: 0 }`
`$reset()`	Reset the state to its initial value	`store.$reset()`
`$patch(partialState)`	Patch the state (object or function)	`store.$patch({ count: 5 })`
`$subscribe(callback, options?)`	Watch state changes	`store.$subscribe((mutation, state) => {...})`
`$dispose()`	Remove store from the pinia instance	`store.$dispose()`
`$onAction(callback)`	Subscribe to actions	`store.$onAction(({ name, after }) => {...})`
`$id`	The unique identifier of the store	`console.log(store.$id)`

State Management

```js
// Access state
const value = store.someState

// Replace state (not recommended)
store.$state = { count: 24 }

// Patch object state
store.$patch({
  count: store.count + 1,
  name: 'Eduardo'
})

// Patch with function (better for multiple changes)
store.$patch((state) => {
  state.count++
  state.items.push({ name: 'new item' })
})
```

Subscription Methods

```
js
```

```
// Subscribe to state changes
const unsubscribe = store.$subscribe((mutation, state) => {
  // 'mutation' is an object containing info about what changed:
  //   - type: How state was changed
  //     - 'direct' - when state properties were directly modified
  //     - 'patch object' - when using $patch with an object
  //     - 'patch function' - when using $patch with a function
  //   - payload: The data that was passed to $patch (undefined for direct mutations)
  //   - storeId: The ID of the store

  // Examples:
  if (mutation.type === 'direct') {
    console.log('Someone directly modified:', state)
    // e.g. store.count++, store.name = 'New name'
  }

  if (mutation.type === 'patch object') {
    console.log('Patched with object:', mutation.payload)
    // e.g. store.$patch({ count: 10, name: 'New name' })
  }

  if (mutation.type === 'patch function') {
    console.log('Patched with function')
    // e.g. store.$patch((state) => { state.count++; state.items.push('new') })
    // Note: payload will be undefined since the function is not available
  }

  // 'state' is the entire current state of the store after the mutation
  localStorage.setItem('cart', JSON.stringify(state))
}, { detached: false }) // detached: true makes it persist after component unmount

// Unsubscribe when no longer needed
unsubscribe()

// Subscribe to actions
const unsubscribeAction = store.$onAction(({
  name,      // Name of the action
  store,     // Store instance
  args,      // Array of parameters passed to the action
  after,     // Hook after the action returns
  onError    // Hook if the action throws
}) => {
  // Before action
```

```
  console.log(`${name} action started with args: ${args.join(', ')}`)

  after((result) => {
    // After action succeeds
    console.log(`${name} action completed with result: ${result}`)
  })

  onError((error) => {
    // If action fails
    console.error(`${name} action failed with error: ${error}`)
  })
})

// Unsubscribe from actions
unsubscribeAction()
```

Store Setup Helper Methods

Method	Description	Example
`defineStore()`	Define a new store	`defineStore('id', options)`
`storeToRefs()`	Extract reactive refs from store state	`const { user, cart } = storeToRefs(store)`
`mapState()`	Map state to computed properties	`...mapState(useStore, ['prop1', 'prop2'])`
`mapActions()`	Map actions to methods	`...mapActions(useStore, ['action1', 'action2'])`
`mapStores()`	Map entire stores	`...mapStores(useUserStore, useCartStore)`
`mapWritableState()`	Map state with get/set	`...mapWritableState(useStore, ['prop1', 'prop2'])`

Common Store Patterns

Using with Components

```js
// In component
import { storeToRefs } from 'pinia'
import { useCounterStore } from '@/stores/counter'

// Setup script
const store = useCounterStore()

// Destructure while keeping reactivity
const { count, doubleCount } = storeToRefs(store)

// Actions can be destructured directly
const { increment } = store
```

Store with TypeScript

ts

```ts
// Define store with TypeScript
export const useUserStore = defineStore('user', {
  state: () => ({
    name: 'Eduardo',
    isAdmin: true,
    likes: ['coding', 'pinia'],
  } as UserState),

  getters: {
    nameInUpperCase(): string {
      return this.name.toUpperCase()
    }
  },

  actions: {
    async fetchUserData(userId: string): Promise<User> {
      // ...
    }
  }
})

// With Composition API + TS
export const useUserStore = defineStore('user', () => {
  const name = ref('')
  const isAdmin = ref(false)

  function setUser(newName: string) {
    name.value = newName
  }

  return { name, isAdmin, setUser }
})
```

Plugins and Extensions

```js
// Add properties to every store
pinia.use(({ store }) => {
  store.$logState = () => {
    console.log(store.$id, JSON.stringify(store.$state))
  }
})

// Plugin with options
pinia.use(createPersistedState({ key: 'persisted' }))
```

Store Composition

```js
// Use stores inside other stores
export const useUserCartStore = defineStore('userCart', () => {
  const user = useUserStore()
  const cart = useCartStore()

  const cartWithUserInfo = computed(() => {
    return {
      items: cart.items,
      user: user.name
    }
  })

  function purchaseItems() {
    cart.purchase(user.id)
  }

  return { cartWithUserInfo, purchaseItems }
})
```