Pinia

생성일: 2026년 2월 11일 오전 11:50

Pinia 핵심 요약 가이드 (Vue.js 상태 관리)

1. Pinia 정의

Vue 컴포넌트 외부에서 **공통된 상태(State)**를 관리하는 공식 저장소(Store).

- **Props Drilling 해결**: 계층 구조가 복잡한 컴포넌트 간 직접적인 데이터 통신 가능.
- **중앙 집중식 관리**: 사용자 정보, 설정값 등 앱 전반에서 공유되는 데이터의 일관성 유지.

2. Store의 3요소 (컴포넌트 구조 대응)

Vue 컴포넌트의 구성 요소와 1:1 대응 구조를 가짐.

1. **State (ref)**: 공유 대상인 **데이터**.
2. **Getters (computed)**: 상태를 가공한 **계산된 값**.
3. **Actions (function)**: 상태 변경 및 비동기 작업을 수행하는 **메서드**.

3. 기본 사용법 및 코드 예제

[Step 1] 스토어 정의하기 (`src/stores/counter.js`)

`defineStore`를 사용하여 저장소의 이름과 로직을 정의합니다.

```
import { defineStore } from 'pinia'
import { ref, computed } from 'vue'

export const useCounterStore = defineStore('counter', () => {
  // 해당 저장소가 가지는 state, getters, actions 를 정의
  // 1. state : ref 함수를 사용
  const count = ref(0)

  // 2. getters : computed 함수를 사용
  const doubleCount = computed(() => count.value * 2)

  // 3. actions : 일반 함수로 정의
  function increment() {
    count.value++
  }

  // return 안에 정의된 항목이 외부에 노출되는 항목
  return { count, doubleCount, increment }
})
```

[Step 2] 컴포넌트에서 사용하기

스토어를 가져와서 함수처럼 호출하여 사용합니다.

1. AppTop.vue (`src/views/AppTop.vue`)

```
<script setup>
import { RouterLink } from "vue-router";
import { useCounterStore } from './stores/counter'
import { storeToRefs } from 'pinia'

const store = useCounterStore()

// 주의: state와 getters는 그냥 구조 분해하면 반응성 해제
// storeToRefs를 사용해야 반응성을 유지
// action을 제외하고 state, getters를 저장소에 반환 : storeToRefs
const { count, doubleCount } = storeToRefs(store)

// actions는 일반 함수이므로 바로 구조 분해.
const { increment } = store
</script>

<template>
  <h1>Top</h1>
  <p>Top</p>
  <h3>총 회원수 : {{ count }}명</h3>
  <p><RouterLink to="/member/list">회원관리는 여기서</RouterLink></p>
</template>
```

2. MemberList.vue (`src/views/member/MemberList.vue`)

```
<script setup>
// 추가 코드
import { useCounterStore } from "@/stores/counter";
const counterStore = useCounterStore();

const findMemberList = async () => {
  const list = await fetch(`${server}/users`)
    .then((res) => res.json())
    .catch((err) => console.log(err));

  //console.log(list);
  members.value = list;
  // 구조분해하지 않고 사용하는 방법
  counterStore.count = members.value.length;
};
</script>
```

4. 핵심 암기 사항

1. **반응성 보존**: State 접근 시 반드시 `storeToRefs()` 사용 권장.
2. **상태 변경 캡슐화**: 데이터 수정은 가급적 스토어 내부 `Action`을 통해 수행(유지보수성 향상).
3. **적소 활용 원칙**: 모든 데이터를 Pinia에 넣지 않음. 로컬 데이터는 기존처럼 `ref`로 관리.

5. 실습 과제: 사용자 스토어 구현

1. `member.js` 스토어 생성 (`src/stores/member.js`)

```
import { ref, computed } from "vue";
import { defineStore } from "pinia";

export const useMemberStore = defineStore("member", () => {

    return {};
}
```

2. 각 컴포넌트에서 공통적인 데이터를 다루는 부분들을 store에 옮기기

    1. MemberList.vue (`src/views/member/MemberList.vue`)

```
const members = ref([]);

const server = "https://jsonplaceholder.typicode.com";
const findMemberList = async () => {
  const list = await fetch(`${server}/users`)
    .then((res) => res.json())
    .catch((err) => console.log(err));

  //console.log(list);
  members.value = list;
};
```

    2. MemberDetail.vue (`src/views/member/MemberDetail.vue`)

```
const member = ref({}); // 회원정보

const server = "https://jsonplaceholder.typicode.com";
const findMemberById = async (memberId) => {
  let info = await fetch(`${server}/users/${memberId}`)
    .then((res) => res.json())
    .catch((err) => console.log(err));
  //console.log(info);
  member.value = info;
};
```

    3. MemberAdd.vue (`src/views/member/MemberAdd.vue`)

```
  let info = await fetch(`${server}/users`, {
        method: "post",
        headers: {
          "content-type": "application/json",
        },
        body: JSON.stringify(member.value),
      })
    .then((res) => res.json())
    .catch((err) => console.log(err));
  let newId = info.id;
```

4. member.js (src/stores/member.js)

```
import { ref, computed } from "vue";
import { defineStore } from "pinia";

const server = "https://jsonplaceholder.typicode.com";

export const useMemberStore = defineStore("member", () => {
  // state
  // 전체 회원 정보
  const members = ref([]);
  // 선택한 회원 정보
  const member = ref({});

  // getters
  // 전체 회원 수
  const count = computed(() => {
    return members.value.length;
  });

  // action
  // 서버에서 전체 회원 정보 가져오기
  const findMemberList = async () => {
    const list = await fetch(`${server}/users`)
      .then((res) => res.json())
      .catch((err) => console.log(err));
    members.value = list;
  };

  // 서버에서 지정한 회원의 상세정보 가져오기
  const findMemberById = async (memberId) => {
    let info = await fetch(`${server}/users/${memberId}`)
      .then((res) => res.json())
      .catch((err) => console.log(err));
    member.value = info;
  };

  // 서버에서 회원 정보 등록하기
  const createMember = async (memberInfo) => {
    let info = await fetch(`${server}/users`, {
```

```
      method: "post",
      headers: {
        "content-type": "application/json",
      },
      body: JSON.stringify(memberInfo),
    })
      .then((res) => res.json())
      .catch((err) => console.log(err));
    return info.id;
  };

  return {
    members,
    member,
    count,
    findMemberList,
    findMemberById,
    createMember,
  };
});
```

3. 각 컴포넌트에서 데이터를 다루는 부분을 store을 이용해서 변경하기

    1. MemberList.vue (`src/views/member/MemberList.vue`)

```
import { useMemberStore } from "@/stores/member";
import { storeToRefs } from "pinia";

const memberStore = useMemberStore();

const { members } = storeToRefs(memberStore);
const { findMemberList } = memberStore;
```

    2. MemberDetail.vue (`src/views/member/MemberDetail.vue`)

```
import { useMemberStore } from "@/stores/member";
import { storeToRefs } from "pinia";

const memberStore = useMemberStore();

const { member } = storeToRefs(memberStore);
const { findMemberById } = memberStore;
```

    3. MemberAdd.vue (`src/views/member/MemberAdd.vue`)

```
```

```
import { useMemberStore } from "@/stores/member";
const memberStore = useMemberStore();

const addMember = async () => {
    // 기존 fetch 함수 대신 store의 action 함수를 호출
  let newId = await memberStore.createMember(member.value);

  router.push({ name: "memberDetail", params: { id: newId } });
};
```

4. AppTop.vue (`src/views/AppTop.vue`)

```
// 기존 코드
import { useCounterStore } from './stores/counter'
import { storeToRefs } from 'pinia'

const store = useCounterStore()

// 주의: state와 getters는 그냥 구조 분해하면 반응성 해제
// storeToRefs를 사용해야 반응성을 유지
// action을 제외하고 state, getters를 저장소에 반환 : storeToRefs
const { count, doubleCount } = storeToRefs(store)

// actions는 일반 함수이므로 바로 구조 분해.
const { increment } = store

// 변경 코드
import { useMemberStore } from "@/stores/member";
import { storeToRefs } from "pinia";
const memberStore = useMemberStore();

const { count } = storeToRefs(memberStore);
```

6. Pinia 상태 지속성 유지 (pinia-plugin-persistedstate)

Pinia의 상태는 페이지 새로고침 시 초기화됨. 이를 로컬 스토리지 등에 자동으로 저장하고 복구하기 위해 플러그인을 사용함.

1. 프러그인 설치

```
npm i pinia-plugin-persistedstate
```

2. 플러그인 설정 (`main.js`)

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";

import { createPinia } from 'pinia'
import piniaPluginPersistedstate from 'pinia-plugin-persistedstate'

const pinia = createPinia()
pinia.use(piniaPluginPersistedstate) // 플러그인 등록

const app = createApp(App);

app.use(pinia)
app.use(router);

app.mount("#app");
```

3. 스토어 적용 예시 (`stores/member.js`)

   스토어 정의 시 `persist: true` 옵션만 추가하면 간단히 적용됨.

```
export const useMemberStore = defineStore('member', () => {
  ...
  return {
    members,
    member,
    count,
    findMemberList,
    findMemberById,
    createMember,
  },
  {
      persist: true // 상태 자동 저장 활성화
    }
);
```