CIS 658 Web Architectures

Pinia II

App State Management

Lecturer: Dr. Yong Zhuang

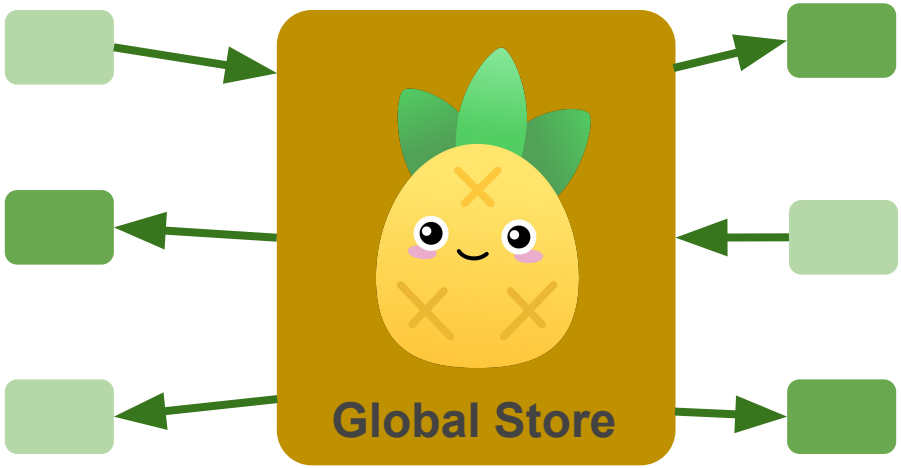
Based on the original version by Daniel Kelly.

Last time

What is state management?

What is Pinia?

- State,



Practice: Pinia(States)

Answer

Pinia Actions

Pinia Actions

ItemStore.ts

```
import { defineStore } from "pinia";
import products from "../data/products.json";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  },

  actions: {
    fill() {},
  },
});
```

Define actions as methods on the actions option

Pinia Actions

UserStore.ts

```ts
import { defineStore } from "pinia";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: [] };
  },

  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
      this.someOtherAction();
    },
    async someOtherAction() {},
  },
});
```

Access state with `this`

Access other actions with `this`

Pinia Actions

user.ts

```ts
type User = {
  name: {
    first: string;
    last: string;
    title: string;
  };
  email: string;
  picture: {
    large: string;
    medium: string;
    thumbnail: string;
  };
};
type RandomUser = {
  results: Array<User>;
};
export type { User, RandomUser };
```

UserStore.ts

```ts
import { defineStore } from "pinia";
import { RandomUser } from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: { results: [] } as RandomUser };
  },

  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
      console.log(this.users);
    },
  },
});
```

Pinia Actions

AnyVueComponent.vue

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();
userStore.fill();
</script>

<template>
  <ul v-for="(u, idx) in userStore.users.results" :key="idx" :u="u">
    <li>{{ u.name.first }}. {{ u.name.last }}</li>
  </ul>
</template>
```

Demo

Pinia Actions

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const { fill } = useUserStore();
fill();
</script>
```

de-structure

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
import { storeToRefs } from "pinia";
const { fill } = storeToRefs(useUserStore());
fill();
</script>
```

won't work

Demo

Pinia Actions: Example

CartStore.ts

```ts
import { defineStore } from "pinia";

export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  actions: {
    addItem(itemId, count) {
      // set the count for the proper item in the state above
    },
  },
});
```

Item.vue

```vue
<button @click="addItem(product.id, $event)">Add Product to Cart</button>
```

Useful for updating state based on user interaction

Pinia Getters

What are Pinia Getters?

Equivalent of computed props on a component

Must explicitly type the return

Access state with `this`

UserStore.ts

```ts
import { defineStore } from "pinia";
import { RandomUser } from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: {} as RandomUser };
  },
  getters: {
    count(): number {
      return this.users.results.length;
    },
  },
  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
    },
  },
});
```

Pinia Getters

Access state with `state`

```
getters: {
  count(state): number {
    return state.users.results.length;
  },
},
```

No need to explicitly type return

```
getters: {
  count: (state) => state.users.results.length,
},
```

single line arrow functions

Access other getters on `this`

```
getters: {
  count: (state) => state.users.results.length,
  doubleCount(): number {
    return this.count * 2;
  },

  findUserByFirstName: (state) => (first: string) => {
    return state.users.results.find(
      (user: User) => user.name.first === first
    );
  },
},
```

Accept arguments by returning a function

Access Getters

as a property

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();
console.log(userStore.count);
</script>
```

de-structure

```
<script setup lang="ts">
import { useUserStore } from "./stores/UserStore";
import { storeToRefs } from "pinia";
const { count } = storeToRefs(useUserStore());
console.log(count.value);
</script>
```

Can de-structure getters from store but must use `storeToRefs`

Demo

Using Stores in Other Stores

CartStore.ts

```ts
import { defineStore } from "pinia";
import { useItemStore } from "./ItemStore";

export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  getters: {
    allProducts() {
      const itemStore = useItemStore();
      return itemStore.products;
    },
  },
  actions: {
    addItem(itemId, count) {
      // set the count for the proper item in the state above
    },
  },
});
```

ItemStore.ts

```ts
import { defineStore } from "pinia";
import products from "../data/products.json";
export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  },
});
```

Use actions from another store in an action

CareStore.ts

```ts
import { defineStore } from "pinia";
import { useItemStore } from "./ItemStore";

export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  actions: {
    reloadProducts() {
      const itemStore = useItemStore();
      return itemStore.fill();
    },
  },
});
```

ItemStore.ts

```ts
import { defineStore } from "pinia";
import products from "../data/products.json";
export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  },
  actions: {
    async fill() {
      //...
    },
  },
});
```

Use actions from another store in an action

CareStore.ts

```
import { defineStore } from "pinia";
import { useItemStore } from "./ItemStore";

const itemStore = useItemStore();
export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  actions: {
    reloadProducts() {
      return itemStore.fill();
    },
  },
});
```

```
▸Uncaught Error: [🍍]: getActivePinia was called with no active Pinia. Did you
forget to install pinia?
    const pinia = createPinia()
    app.use(pinia)
This will fail in production.
    at useStore (pinia.esm-browser.js:1638:19)
```

Subscribe to Pinia Stores

Notify me when something changes in this store, so I can update the UI or perform other actions in response.

- Watch state for changes.
- Monitor actions for calls.
- Perform side effects.
- Measure how long your actions take to run.
- Trigger user notifications.
- Log errors to third-party services.

Subscribe to actions

AnyVueComponent.vue

```
unsubscribe = userStore.$onAction(
  ({
    name, // name of the action
    store, // store instance, same as `someStore`
    args, // array of parameters passed to the action
    after, // hook after the action returns or resolves
    onError, // hook if the action throws or rejects
  }) => {
    // Action-related logic code here...
  }
);
```

Use conditional to run on select actions

AnyVueComponent.vue

```
unsubscribe = userStore.$onAction(
  ({
    name, // name of the action
    store, // store instance, same as `someStore`
    args, // array of parameters passed to the action
    after, // hook after the action returns or resolves
    onError, // hook if the action throws or rejects
  }) => {
    if (name === "fill") {
      // more code
    }
  }
);
```

After and onError

UserStore.ts

```ts
import { defineStore } from "pinia";
import { RandomUser} from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: {} as RandomUser };
  },

  getters: {
    count: (state) => state.users.results.length,
  },
  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
      );
      this.users = await res.json();
      return this.count;
    },
  },
});
```

AnyVueComponent.vue

```ts
unsubscribe = userStore.$onAction(
  ({
    name, // name of the action
    store, // store instance, same as `someStore`
    args, // array of parameters passed to the action
    after, // hook after the action returns or resolves
    onError, // hook if the action throws or rejects
  }) => {
    const startTime = Date.now();
    // this will trigger before an action on `store` is executed
    console.log(`Start "${name}" with params [${args.join(", ")}].`);
    if (name === "fill") {
      //  after() will trigger if the action succeeds and after it has fully run.
      after((result) => {
        console.log(
          `Finished "${name}" after ${
            Date.now() - startTime
          }ms.\nResult: ${result}.`
        );
      });
      // onError() will trigger if the action throws or returns a promise that rejects
      onError((error) => {
        console.warn(
          `Failed "${name}" after ${
            Date.now() - startTime
          }ms.\nError: ${error}.`
        );
      });
    }
  }
);
```

Demo

Subscribe to the state

AnyVueComponent.vue

```
onMounted(() => {
  userStore.$subscribe((mutation, state) => {
    // 'direct' | 'patch object' | 'patch function'
    mutation.type;

    // same as userStore.$id
    mutation.storeId;

    // only available with mutation.type === 'patch object'
    mutation.payload; // patch object passed to $patch()
  });
});
```

Demo

Online Doc

The 'Patch Object' in Pinia

The $patch() method is a convenient way to apply multiple state changes at once. It simplifies state management by allowing batch updates.

- When to use: Handling complex state structures or when multiple state properties need to be updated simultaneously.
- Value Setting:
  - For each key in the patch object, the value is the new value that you want to set for that corresponding state property.
  - The $patch() method ensures that these updates are reactive and efficiently managed by Vue's reactivity system.

The 'Patch Object' in Pinia

```ts
import { defineStore } from "pinia";

export const useUserStore = defineStore("userStore", {
  state: () => ({
    name: "",
    age: 0,
    email: "",
  }),
  // actions, getters, etc.
});
```

UserStore.ts

```vue
<script setup lang="ts">
import { onMounted } from "vue";
import { useUserStore } from "./stores/UserStore";
const userStore = useUserStore();

onMounted(() => {
  userStore.$subscribe((mutation, state) => {
    if (mutation.type === "patch object") {
      console.log("Patch object:", mutation.payload);
    }
  });
  userStore.$patch({
    name: "John Doe",
    email: "john@example.com",
  });
});
</script>

<template>
  <ul>
    <li>{{ userStore.name }}</li>
    <li>{{ userStore.age }}</li>
    <li>{{ userStore.email }}</li>
  </ul>
</template>
```

AnyVueComponent.vue

Demo

Online Doc

Exercise from the previous class

Complete the store-app implementation with the following functionalities:

- Implement a CartStore using Pinia to temporarily store the user's selected items for checkout.
- Utilize the $patch method to add items into cart.
- Displays all items currently in the cart.

store-app

Persist data?

Refresh-proof your Pinia Stores