Pinia

Pinia源码part3-Store的创建createOptionsStore方法

上一节，讲解了defineStore的内部实现，defineStore里面有牵涉store创建的逻辑。这一章就来讲述一下store创建的逻辑和实现。

```
if (!pinia._s.has(id)) {
  // creating the store registers it in `pinia._s`
  if (isSetupStore) {
    createSetupStore(id, setup, options, pinia)
  } else {
    createOptionsStore(id, options as any, pinia)
  }

  /* istanbul ignore else */
  if (__DEV__) {
    // @ts-expect-error: not the right inferred type
    useStore._pinia = pinia
  }
}
```

上面的代码来自于上一节。store通过id区分不同store的数据。如果当前传入的id与现有的所有store id都不符合，那么就表示之前没有给这个id创建数据，则需要开始创建，代码里面使用了createSetupStore 与 createOptionsStore 这2个方法创建。

> 其实即使使用createOptionsStore常见store数据，内部同样会调用createSetupStore. 后面的源码分析会讲解。

那么使用哪个方法创建取决于defineStore 传入的参数类型。isSetupStore 决定了使用了哪个方法创建，那么先来了解一下isSetupStore,

```
function defineStore(
  // TODO: add proper types from above
  idOrOptions: any,
  setup?: any,
  setupOptions?: any
): StoreDefinition {
  // ... 省略其他代码
  const isSetupStore = typeof setup === 'function'
}
```

那么可以知道isSetupStore的值是通过setup是否是函数来决定的, 而setup作为defineStore的第二个参数。说到这里，是否回忆起第一章提到会有多种创建store的方法，第一章就讲解了2个。这里又出现了第三个。

如果setup是一个函数，那么创建store数据的同时，会运行setup函数然后获取返回值让其挂在pinia实例上面成为pinia的一个属性。后面的源码会分析具体实现方法，这里先稍微理解其内部的实现目的，即为什么这样设计。其设计的目的就是扩展pinia的功能。

现在进入createOptionsStore函数，

```
function createOptionsStore<
  Id extends string,
  S extends StateTree,
  G extends _GettersTree<S>,
  A extends _ActionsTree
>(
  id: Id,
  options: DefineStoreOptions<Id, S, G, A>,
  pinia: Pinia,
  hot?: boolean
): Store<Id, S, G, A> {
  const { state, actions, getters } = options

  const initialState: StateTree | undefined = pinia.state.value[id]

  let store: Store<Id, S, G, A>

  function setup() {
    if (!initialState && (!__DEV__ || !hot)) {
      /* istanbul ignore if */
      if (isVue2) {
        set(pinia.state.value, id, state ? state() : {})
      } else {
        pinia.state.value[id] = state ? state() : {}
      }
    }

    // avoid creating a state in pinia.state.value
    const localState =
      __DEV__ && hot
        ? // use ref() to unwrap refs inside state TODO: check if this is still necessary
          toRefs(ref(state ? state() : {}).value)
        : toRefs(pinia.state.value[id])

    return assign(
      localState,
      actions,
      Object.keys(getters || {}).reduce((computedGetters, name) => {
        computedGetters[name] = markRaw(
          computed(() => {
            setActivePinia(pinia)
            // it was created just before
            const store = pinia._s.get(id)!

            // allow cross using stores
            /* istanbul ignore next */
```

```
            if (isVue2 && !store._r) return

            // @ts-expect-error
            // return getters![name].call(context, context)
            // TODO: avoid reading the getter while assigning with a global variable
            return getters![name].call(store, store)
          })
        )
        return computedGetters
      }, {} as Record<string, ComputedRef>)
    )
  }

  store = createSetupStore(id, setup, options, pinia, hot)

  store.$reset = function $reset() {
    const newState = state ? state() : {}
    // we use a patch to group all changes into one single subscription
    this.$patch(($state) => {
      assign($state, newState)
    })
  }

  return store as any
}
```

一点一点来分析，

```
const { state, actions, getters }
= options

const initialState: StateTree | undefined
= pinia.state.value[id]

let store: Store<Id, S, G, A>
```

1. 首先从defineStore函数传进来的options解构出来 state, actions, getters。
2. 然后尝试获取pinia实例上面的state属性存储的与传入store id相匹配的数据。当然，第一次进来因为数据还没有，所以会返回undefined。（对state 这个属性如果没有印象，则返回第一章，里面有讲解, state就是一个ref响应式对象）
3. 创建一个store变量，接下来使用store存储数据。

接下来setup 函数，

```
function setup() {
    if (!initialState && (!__DEV__ || !hot)) {
      /* istanbul ignore if */
      if (isVue2) {
        set(pinia.state.value, id, state ? state() : {})
```

```
      } else {
        pinia.state.value[id] = state ? state() : {}
      }
    }

    // avoid creating a state in pinia.state.value
    const localState =
      __DEV__ && hot
        ? // use ref() to unwrap refs inside state TODO: check if this is still necessary
          toRefs(ref(state ? state() : {}).value)
        : toRefs(pinia.state.value[id])

    return assign(
      localState,
      actions,
      Object.keys(getters || {}).reduce((computedGetters, name) => {
        computedGetters[name] = markRaw(
          computed(() => {
            setActivePinia(pinia)
            // it was created just before
            const store = pinia._s.get(id)!

            // allow cross using stores
            /* istanbul ignore next */
            if (isVue2 && !store._r) return

            // @ts-expect-error
            // return getters![name].call(context, context)
            // TODO: avoid reading the getter while assigning with a global variable
            return getters![name].call(store, store)
          })
        )
        return computedGetters
      }, {} as Record<string, ComputedRef>)
    )
  }
```

1. 获取上一步解构出来的state，然后运行收集返回结果，然后添加到，pinia.state.value[id]
2. 如果当前开发环境，而且需要热更新，则直接从传入的option里面获取state，然后toRefs转换成一般对象，其每一个属性都是一个ref。这么做为了避免对pinia.state.value[id]进行修改。
3. 如果不是开发环境，则直接从pinia.state.value[id] 获取，然后toRefs 转换成一个一般对象，其每一个属性都是一个ref。
4. 然后通过Object.assign 让 localState, actions, getters进行融合，生成一个新的大对象。
5. 最后讲一下getter的转换过程，使用了reduce遍历getter，对每一个getter进行computed包裹，其内部就是调用getter，而且通过当前活跃的pinia获取其内部的对应的id的store数据，然后让其作为getter的第一个参数让每一个getter内部都可以通过store获取store上面的任意数据。
6. getter转换会生成一个新的对象computedGetters, 返回找个对象。

> 总结一下setup函数其实做了
>
> 1. 传入的state全部转换成响应式对象
> 2. getter的转换，包裹computed然后配置store以提供getter内部对store数据的获取

创建好的setup函数至此也只是定义完成，而没有被调用，

```
store = createSetupStore(
id, setup, options, pinia, hot)
```

然后createOptionsStore内部调用createSetupStore函数，内部的实现将会在下一章讲解。这里可以知道，之前的setup函数作为了createSetupStore的参数

接下来$reset函数，

```
store.$reset = function $reset() {
    const newState = state ? state() : {}
    // we use a patch to group all changes into one single subscription
    this.$patch(($state) => {
      assign($state, newState)
    })
  }

  return store as any
```

reset函数就是通过$patch函数对现有的$state变量融合，$patch函数与$state属性都在下一章的createSetupStore函数定义。

> $state函数其实就是通过Object.defineProperty给store的一个属性其返回pinia.state.value[$id]