Pinia

Pinia源码part4-Store的创建(2) - createSetupsStore方法

上一章讲到了createOptionsStore内部也是调用了createSetupStore方法。

createSetupStore方法比较长，会一段一段的慢慢来讲解。(store.ts, 179行开始)

```
function createSetupStore<
  Id extends string,
  SS,
  S extends StateTree,
  G extends Record<string, _Method>,
  A extends _ActionsTree
>(
  $id: Id,
  setup: () => SS,
  options:
    | DefineSetupStoreOptions<Id, S, G, A>
    | DefineStoreOptions<Id, S, G, A> = {},
  pinia: Pinia,
  hot?: boolean
): Store<Id, S, G, A> {
  let scope!: EffectScope
  const buildState = (options as DefineStoreOptions<Id, S, G, A>).state

  const optionsForPlugin: DefineStoreOptionsInPlugin<Id, S, G, A> = assign(
    { actions: {} as A },
    options
  )
```

5个参数，

1. $id - store id
2. setup - 函数创建store的逻辑，根据用户业务需要可以自定义，方便扩展，
3. options，store的数据比如state等等。
4. pinia，Pinia的实例。
5. hot，是否需要热更新。

buildState - 就是store 的options

optionsForPlugin, 后面会讲解，

scope, 就是EffectScope 控制响应式副作用组件之外的取消等等

```
/* istanbul ignore if */
```

```
  if (__DEV__ && !pinia._e.active) {
    throw new Error('Pinia destroyed')
  }

  // watcher options for $subscribe
  const $subscribeOptions: WatchOptions = {
    deep: true,
    // flush: 'post',
  }
```

1. 如果开发环境，pinia._e 的effectScope不处于active，那么就直接报错，pinia 已经销毁。第一章里面，已经effectScope(true),所以active肯定有值。
2. $subscribeOptions - 后面给vue watcher一个配置项。

```
/* istanbul ignore else */
  if (__DEV__ && !isVue2) {
    $subscribeOptions.onTrigger = (event) => {
      /* istanbul ignore else */
      if (isListening) {
        debuggerEvents = event
        // avoid triggering this while the store is being built and the state is being set
      } else if (isListening == false && !store._hotUpdating) {
        // let patch send all the events together later
        /* istanbul ignore else */
        if (Array.isArray(debuggerEvents)) {
          debuggerEvents.push(event)
        } else {
          console.error(
            '  debuggerEvents should be an array. This is most likely an internal Pinia bu
          )
        }
      }
    }
  }
```

1. 方便开发者调试，添加了追踪subscription的进度的接口，可以给开发者自定义追踪的逻辑。

```
// internal state
  let isListening: boolean // set to true at the end
  let isSyncListening: boolean // set to true at the end
  let subscriptions: SubscriptionCallback<S>[] = markRaw([])
  let actionSubscriptions: StoreOnActionListener<Id, S, G, A>[] = markRaw([])
  let debuggerEvents: DebuggerEvent[] | DebuggerEvent
  const initialState = pinia.state.value[$id] as UnwrapRef<S> | undefined

// avoid setting the state for option stores are it is set
  // by the setup
  if (!buildState && !initialState && (!__DEV__ || !hot)) {
    /* istanbul ignore if */
  if (isVue2) {
   set(pinia.state.value, $id, {})
  } else {
   pinia.state.value[$id] = {}
  }
 }
 const hotState = ref({} as S)
```

1. 开发者如果没有提供buildState, 也没有任何当前store 状态数据则进行store状态数据的初始化，一个对象。

```
function $patch(stateMutation: (state: UnwrapRef<S>) => void): void
function $patch(partialState: _DeepPartial<UnwrapRef<S>>): void
function $patch(
  partialStateOrMutator:
   | _DeepPartial<UnwrapRef<S>>
   | ((state: UnwrapRef<S>) => void)
 ): void {
  let subscriptionMutation: SubscriptionCallbackMutation<S>
  isListening = isSyncListening = false
  // reset the debugger events since patches are sync
  /* istanbul ignore else */
  if (__DEV__) {
   debuggerEvents = []
  }
  if (typeof partialStateOrMutator === 'function') {
   partialStateOrMutator(pinia.state.value[$id] as UnwrapRef<S>)
   subscriptionMutation = {
    type: MutationType.patchFunction,
    storeId: $id,
    events: debuggerEvents as DebuggerEvent[],
   }
  } else {
   mergeReactiveObjects(pinia.state.value[$id], partialStateOrMutator)
   subscriptionMutation = {
    type: MutationType.patchObject,
    payload: partialStateOrMutator,
    storeId: $id,
    events: debuggerEvents as DebuggerEvent[],
   }
  }
  nextTick().then(() => {
   isListening = true
  })
  isSyncListening = true
  // because we paused the watcher, we need to manually call the subscriptions
  triggerSubscriptions(
   subscriptions,
   subscriptionMutation,
   pinia.state.value[$id] as UnwrapRef<S>
  )
}
```

pinia的$patch方法，用来修改state，对比action，其可以一次性的提供所有需要改的部分，然后统一更新给原来的state。其可以接受一个对象或者函数。那么一步一步来理解一下。

1. 如果参数是一个函数，把此次需要更新的数据等信息组合成一个对象赋值给subscriptionMutation,
2. 如果参数是一个对象， 和第一步一样，只不过多了一个payload属性，里面是此次更新的数据。
3. 然后调用nextTick promise让 isListening 状态在下一个微任务变成true。即异步更改状态
4. 设置isSyncListening 的值 true，
5. 手动调用所有的subscription，让这次更新的数据等信息subscriptionMutation作为第二个参数传入
6. 上面的 3，4，5 会在pinia的$subscription章节深入讲解其作用

```
/* istanbul ignore next */
const $reset = __DEV__
  ? () => {
      throw new Error(
        ` : Store "${$id}" is build using the setup syntax and does not implement $reset
      )
    }
  : noop

function $dispose() {
  scope.stop()
  subscriptions = []
  actionSubscriptions = []
  pinia._s.delete($id)
}
```

1. 设置$reset变量，其作用就是占位符。最后会在createOptionsStore里面实现真正的$reset。
2. $dispose函数就是清理当前id 对应的store的所有数据，比如删除state, subscription, 以及停止所有响应式对象的副作用。

```
/**
 * Wraps an action to handle subscriptions.
 *
 * @param name - name of the action
 * @param action - action to wrap
 * @returns a wrapped action to handle subscriptions
 */
function wrapAction(name: string, action: _Method) {
  return function (this: any) {
    setActivePinia(pinia)
    const args = Array.from(arguments)

    const afterCallbackList: Array<(resolvedReturn: any) => any> = []
    const onErrorCallbackList: Array<(error: unknown) => unknown> = []
    function after(callback: _ArrayType<typeof afterCallbackList>) {
      afterCallbackList.push(callback)
    }
    function onError(callback: _ArrayType<typeof onErrorCallbackList>) {
      onErrorCallbackList.push(callback)
    }

    // @ts-expect-error
    triggerSubscriptions(actionSubscriptions, {
      args,
      name,
      store,
      after,
      onError,
    })

    let ret: any
    try {
      ret = action.apply(this && this.$id === $id ? this : store, args)
      // handle sync errors
    } catch (error) {
      triggerSubscriptions(onErrorCallbackList, error)
      throw error
    }

    if (ret instanceof Promise) {
      return ret
        .then((value) => {
          triggerSubscriptions(afterCallbackList, value)
          return value
        })
        .catch((error) => {
          triggerSubscriptions(onErrorCallbackList, error)
          return Promise.reject(error)
        })
    }

    // allow the afterCallback to override the return value
    triggerSubscriptions(afterCallbackList, ret)
    return ret
  }
}
```

1. 此函数用来处理store 数据里面的所有getter ，actions 等函数的各种不同的subscription，比如 onaction, afterCallback 等等 。
2. afterCallbacklist, onErrorCallbackList 提供给after，onError函数作为容器使用的，这2个函数会作为参数传给triggerSubscription ，其作用就是让开发者可以自定义subscription，onerror 的自定义逻辑。比如可以log当前的操作等任何开发者想要完成的。
3. 执行当前的函数，如果当前函数是一个promise，则使用then处理异步，确保一定可以返回。也就是说，完全可以在getter 或者 action 返回promise，pinia底层会帮住开发者处理异步逻辑确保一定会有返回值以

及对异步错误的处理。

4. 调用用户自定义afterCallbacklist的callback
5. 如果不是promise直接返回当前函数的值。

```
const partialStore = {
    _p: pinia,
    // _s: scope,
    $id,
    $onAction: addSubscription.bind(null, actionSubscriptions),
    $patch,
    $reset,
    $subscribe(callback, options = {}) {
      const removeSubscription = addSubscription(
        subscriptions,
        callback,
        options.detached,
        () => stopWatcher()
      )
      const stopWatcher = scope.run(() =>
        watch(
          () => pinia.state.value[$id] as UnwrapRef<S>,
          (state) => {
            if (options.flush === 'sync' ? isSyncListening : isListening) {
              callback(
                {
                  storeId: $id,
                  type: MutationType.direct,
                  events: debuggerEvents as DebuggerEvent,
                },
                state
              )
            }
          },
          assign({}, $subscribeOptions, options)
        )
      )!

      return removeSubscription
    },
    $dispose,
  } as _StoreWithState<Id, S, G, A>
```

1. partialStore函数处理store里面的部分数据。是的，部分数据，pinia底层对于store的数据处理分成2步，除了上面函数，还有在createOptionsStore 里面处理。
2. partialStore, 处理store 数据里面的$id, $onAction, $patch,$reset,$subscription,$dispose 等。除了这部分数据,partialStore没有处理的就是state, getter, action等等。
3. $onAction 就是addSubscription函数，其内部实现会放在下一章的subscription讲解
4. 剩下的部分比如$reset, $dispose, 等等已经在前面讲解

```
const store: Store<Id, S, G, A> = reactive(
    assign(
      __DEV__ && IS_CLIENT
        ? // devtools custom properties
          {
            _customProperties: markRaw(new Set<string>()),
            _hmrPayload,
          }
        : {},
      partialStore
      // must be added later
      // setupStore
    )
  ) as unknown as Store<Id, S, G, A>
```

1. 刚刚创建好的partialStore与customProperties融合成一个对象，然后转换成reactivity响应式
2. customProperties就是第一章讲解的pinia 可以让开发者自定义plugin，plugin 返回一个对象，对象的属性容器就是customProperties.

```
pinia._s.set($id, store)
```

1. 把store放在pinia实例上面，第一章对pinia的各个属性都有讲解。其就是一个map，所以这里使用set添加(key，val)
2. 正如上面所说partialStore不是完全的store数据。所以直到这里，store 也只是处理了部分数据(partialStore)

```
const setupStore = pinia._e.run(() => {
    scope = effectScope()
    return scope.run(() => setup())
  })

```

1. 调用setup函数，这个函数就是createSetupStore的一个参数，它可以来自于2个地方
2. 第一个可能来自于用户创建store会调用defineStore(第二章有讲解)，如果defineStore第二个参数存在而且类型是一个函数，那么就是setup 函数。
3. 第二个可能来自于createOptionsStore内部pinia自己的setup 函数。createOptionStore内部使用createSetupStore函数，所以如果用户没有提供, 就会使用pinia 提供的setup 函数。
4. 运行setup函数的返回值保存在setupStore变量上面,
5. setup函数处理了partialStore没有处理的store数据的其他部分，其设计的目的提供了开发者更高灵活的接口提供开发者可混合pinia内置状态的可能，强！。

```
  // overwrite existing actions to support $onAction
  for (const key in setupStore) {
```

```
  const prop = setupStore[key]

  if ((isRef(prop) && !isComputed(prop)) || isReactive(prop)) {
    // mark it as a piece of state to be serialized
    if (__DEV__ && hot) {
      set(hotState.value, key, toRef(setupStore as any, key))
      // createOptionStore directly sets the state in pinia.state.value so we
      // can just skip that
    } else if (!buildState) {
      // in setup stores we must hydrate the state and sync pinia state tree with the re
      if (initialState && shouldHydrate(prop)) {
        if (isRef(prop)) {
          prop.value = initialState[key]
        } else {
          // probably a reactive object, lets recursively assign
          mergeReactiveObjects(prop, initialState[key])
        }
      }
      // transfer the ref to the pinia state to keep everything in sync
      /* istanbul ignore if */
      if (isVue2) {
        set(pinia.state.value[$id], key, prop)
      } else {
        pinia.state.value[$id][key] = prop
      }
    }

    /* istanbul ignore else */
    if (__DEV__) {
      _hmrPayload.state.push(key)
    }
    // action
  } else if (typeof prop === 'function') {
    // @ts-expect-error: we are overriding the function we avoid wrapping if
    const actionValue = __DEV__ && hot ? prop : wrapAction(key, prop)
    // this a hot module replacement store because the hotUpdate method needs
    // to do it with the right context
    /* istanbul ignore if */
    if (isVue2) {
      set(setupStore, key, actionValue)
    } else {
      // @ts-expect-error
      setupStore[key] = actionValue
    }

    /* istanbul ignore else */
    if (__DEV__) {
      _hmrPayload.actions[key] = prop
    }

    // list actions so they can be used in plugins
    // @ts-expect-error
    optionsForPlugin.actions[key] = prop
  } else if (__DEV__) {
    // add getters for devtools
    if (isComputed(prop)) {
      _hmrPayload.getters[key] = buildState
        ? // @ts-expect-error
          options.getters[key]
        : prop
      if (IS_CLIENT) {
        const getters: string[] =
          // @ts-expect-error: it should be on the store
          setupStore._getters || (setupStore._getters = markRaw([]))
        getters.push(key)
      }
    }
  }
}
```

1. setup函数运行返回的结果进行进一步处理，
2. 遍历返回结果，如果当前对象的属性值是ref而且不是计算属性,或者reactive。当前环境开发环境，需要热更新，则使用set热更新。如果不是开发环境或者不需要热更新，则判断是否用户自己提供了buildState(如果使用defineStore创建store提供了第三个参数)。如果没有提供第三个参数，则判断当前的属性值是否可以进行混合和initialState。如果需要和initialState混合，而且当前的属性值是ref响应式，则让initialState的相同属性值覆盖用户传入的值。如果不需要混合，则认为当前对象是reactivity类型，进行递归融合，然后处理好的prop挂在pinia state上，所有pinia state都可以给用户直接使用。
3. 如果当前的属性值是函数，则使用上面的wrapaction对函数进行subscription的处理。
4. 如果也不是函数，而且开发环境，则添加在devtools的getter。

```
// add the state, getters, and action properties
  /* istanbul ignore if */
  if (isVue2) {
    Object.keys(setupStore).forEach((key) => {
      set(
        store,
        key,
        // @ts-expect-error: valid key indexing
        setupStore[key]
      )
    })
  } else {
    assign(store, setupStore)
    // allows retrieving reactive objects with `storeToRefs()`. Must be called after assig
    // Make `storeToRefs()` work with `reactive()` #799
    assign(toRaw(store), setupStore)
  }
```

1. 让2个部分的store数据融合，成为完整的store数据，
2. 然后让融合的全部数据toRaw获取代理对象的原始值(store之前使用了reactivity进行包裹了。)

```
// use this instead of a computed with setter to be able to create it anywhere
 // without linking the computed lifespan to wherever the store is first
 // created.
 Object.defineProperty(store, '$state', {
   get: () => (__DEV__ && hot ? hotState.value : pinia.state.value[$id]),
   set: (state) => {
     /* istanbul ignore if */
     if (__DEV__ && hot) {
       throw new Error('cannot set hotState')
     }
     $patch(($state) => {
       assign($state, state)
     })
   },
 })
```

1. 给store添加属性$state，方便开发者直接使用$state对数据进行修改而不需要使用setter
2. 内部就是调用$patch实现这个功能，$path 函数已经在上面讲解。

```
// add the hotUpdate before plugins to allow them to override it
 /* istanbul ignore else */
 if (__DEV__) {
   store._hotUpdate = markRaw((newStore) => {
     store._hotUpdating = true
     newStore._hmrPayload.state.forEach((stateKey) => {
       if (stateKey in store.$state) {
         const newStateTarget = newStore.$state[stateKey]
         const oldStateSource = store.$state[stateKey]
         if (
           typeof newStateTarget === 'object' &&
           isPlainObject(newStateTarget) &&
           isPlainObject(oldStateSource)
         ) {
           patchObject(newStateTarget, oldStateSource)
         } else {
           // transfer the ref
           newStore.$state[stateKey] = oldStateSource
         }
       }
       // patch direct access properties to allow store.stateProperty to work as
       // store.$state.stateProperty
       set(store, stateKey, toRef(newStore.$state, stateKey))
     })

     // remove deleted state properties
     Object.keys(store.$state).forEach((stateKey) => {
       if (!(stateKey in newStore.$state)) {
         del(store, stateKey)
       }
```

```
      })

      // avoid devtools logging this as a mutation
      isListening = false
      isSyncListening = false
      pinia.state.value[$id] = toRef(newStore._hmrPayload, 'hotState')
      isSyncListening = true
      nextTick().then(() => {
        isListening = true
      })

      for (const actionName in newStore._hmrPayload.actions) {
        const action: _Method = newStore[actionName]

        set(store, actionName, wrapAction(actionName, action))
      }

      // TODO: does this work in both setup and option store?
      for (const getterName in newStore._hmrPayload.getters) {
        const getter: _Method = newStore._hmrPayload.getters[getterName]
        const getterValue = buildState
          ? // special handling of options api
            computed(() => {
              setActivePinia(pinia)
              return getter.call(store, store)
            })
          : getter

        set(store, getterName, getterValue)
      }

      // remove deleted getters
      Object.keys(store._hmrPayload.getters).forEach((key) => {
        if (!(key in newStore._hmrPayload.getters)) {
          del(store, key)
        }
      })

      // remove old actions
      Object.keys(store._hmrPayload.actions).forEach((key) => {
        if (!(key in newStore._hmrPayload.actions)) {
          del(store, key)
        }
      })

      // update the values used in devtools and to allow deleting new properties later on
      store._hmrPayload = newStore._hmrPayload
      store._getters = newStore._getters
      store._hotUpdating = false
    })

    const nonEnumerable = {
      writable: true,
      configurable: true,
      // avoid warning on devtools trying to display this property
      enumerable: false,
    }

    if (IS_CLIENT) {
      // avoid listing internal properties in devtools
      ;(
        ['_p', '_hmrPayload', '_getters', '_customProperties'] as const
      ).forEach((p) => {
        Object.defineProperty(store, p, {
          value: store[p],
          ...nonEnumerable,
        })
      })
    }
  }
```

1. 上面的代码给hrm使用，有兴趣的可以自己阅读

```
// apply all plugins
 pinia._p.forEach((extender) => {
   /* istanbul ignore else */
   if (__DEV__ && IS_CLIENT) {
     const extensions = scope.run(() =>
       extender({
         store,
         app: pinia._a,
         pinia,
         options: optionsForPlugin,
       })
     )!
     Object.keys(extensions || {}).forEach((key) =>
       store._customProperties.add(key)
     )
     assign(store, extensions)
   } else {
     assign(
       store,
       scope.run(() =>
         extender({
           store,
           app: pinia._a,
           pinia,
           options: optionsForPlugin,
         })
       )!
     )
   }
 })
```

1. 对用户自定义的pinia的plugin进行处理，依次调用
2. 返回值一一挂在pinia store 上面。

```
if (
    __DEV__ &&
    store.$state &&
    typeof store.$state === 'object' &&
    typeof store.$state.constructor === 'function' &&
    !store.$state.constructor.toString().includes('[native code]')
  ) {
    console.warn(
      `[ ]: The "state" must be a plain object. It cannot be\n` +
        `\tstate: () => new MyClass()\n` +
        `Found in store "${store.$id}".`
    )
  }
```

1. 如果state的不是一个普通对象，则直接报错。

```
// only apply hydrate to option stores with an initial state in pinia
  if (
    initialState &&
    buildState &&
    (options as DefineStoreOptions<Id, S, G, A>).hydrate
  ) {
    ;(options as DefineStoreOptions<Id, S, G, A>).hydrate!(
      store.$state,
      initialState
    )
  }
```

1. ssr 使用，如果有机会讲解ssr会回来讲解。

```
isListening = true
isSyncListening = true
return store
```

1. 让subscription的所有状态都转为true
2. 返回处理好的store