一文搞懂pinia状态管理（保姆级教程）

前言

Vue3已经推出很长时间了，它周边的生态也是越来越完善了。之前我们使用Vue2的时候，Vuex可以说是必备的，它作为一个状态管理工具，给我们带来了极大的方便。Vue3推出后，虽然相对于Vue2很多东西都变了，但是核心的东西还是没有变的，比如说状态管理、路由等等。再Vue3种，尤大神推荐我们使用pinia来实现状态管理，他也说pinia就是Vuex的新版本。

那么pinia究竟是何方神圣，本篇文章带大家一起学透它！

1.pinia是什么?

如果你学过Vue2，那么你一定使用过Vuex。我们都知道Vuex在Vue2中主要充当状态管理的角色，所谓状态管理，简单来说就是一个存储数据的地方，存放在Vuex中的数据在各个组件中都能访问到，它是Vue生态中重要的组成部分。

既然Vuex那么重要，那么在Vue3中岂能丢弃！

在Vue3中，可以使用传统的Vuex来实现状态管理，也可以使用最新的pinia来实现状态管理，我们来看看官网如何解释pinia的。

官网解释：

> Pinia 是 Vue 的存储库，它允许您跨组件/页面共享状态。

从上面官网的解释不难看出，pinia和Vuex的作用是一样的，它也充当的是一个存储数据的作用，存储在pinia的数据允许我们在各个组件中使用。

实际上，pinia就是Vuex的升级版，官网也说过，为了尊重原作者，所以取名pinia，而没有取名Vuex，所以大家可以直接将pinia比作为Vue3的Vuex。

2.为什么要使用pinia?

很多小伙伴内心是抗拒学习新东西的，比如我们这里所说的pinia，很多小伙伴可能就会抛出一系列的疑问：为什么要学习pinia？pinia有什么优点吗？既然Vue3还能使用Vuex为什么我还要学它？.....

针对上面一系列的问题，我相信很多刚开始学习pinia的小伙伴都会有，包括我自己当初也有这个疑问。当然，这些问题其实都有答案，我们不可能平白无故的而去学习一样东西吧！肯定它有自己的优点的，所以我们这里先给出pinia的优点，大家心里先有个大概，当你熟练使用它之后，在会过头来看这些优点，相信你能理解。

优点：

- Vue2和Vue3都支持，这让我们同时使用Vue2和Vue3的小伙伴都能很快上手。
- pinia中只有state、getter、action，抛弃了Vuex中的Mutation，Vuex中mutation一直都不太受小伙伴们的待见，pinia直接抛弃它了，这无疑减少了我们工作量。
- pinia中action支持同步和异步，Vuex不支持
- 良好的Typescript支持，毕竟我们Vue3都推荐使用TS来编写，这个时候使用pinia就非常合适了
- 无需再创建各个模块嵌套了，Vuex中如果数据过多，我们通常分模块来进行管理，稍显麻烦，而pinia中每个store都是独立的，互相不影响。
- 体积非常小，只有1KB左右。
- pinia支持插件来扩展自身功能。
- 支持服务端渲染。

pinia的优点还有非常多，上面列出的主要是它的一些主要优点，更多细节的地方还需要大家在使用的时候慢慢体会。

3.准备工作

想要学习pinia，最好有Vue3 的基础，明白组合式API是什么。如果你还不会Vue3，建议先去学习Vue3。

本篇文章讲解pinia时，全部基于Vue3来讲解，至于Vue2中如何使用pinia，小伙伴们可以自行去pinia官网学习，毕竟Vue2中使用pinia的还是少数。

项目搭建：

我们这里搭建一个最新的Vue3 + TS + Vite项目。

执行命令：

```
npm create vite@latest my-vite-app --template vue-ts
```

运行项目：

```
npm install
npm run dev
```

删除app.vue中的其它无用代码，最终页面如下：

4.pinia基础使用

4.1 安装pinia

和vue-router、vuex等一样，我们想要使用pinia都需要先安装它，安装它也比较简单。

安装命令：

```
yarn add pinia
# 或者使用 npm
npm install pinia
```

安装完成后我们需要将pinia挂载到Vue应用中，也就是我们需要创建一个根存储传递给应用程序，简单来说就是创建一个存储数据的数据桶，放到应用程序中去。

修改main.js，引入pinia提供的createPinia方法，创建根存储。

代码如下：

```
// main.ts

import { createApp } from "vue";
import App from "./App.vue";
import { createPinia } from "pinia";
const pinia = createPinia();

```

```
const app = createApp(App);
app.use(pinia);
app.mount("#app");
```

4.2 创建store

store简单来说就是数据仓库的意思，我们数据都放在store里面。当然你也可以把它理解为一个公共组件，只不过该公共组件只存放数据，这些数据我们其它所有的组件都能够访问且可以修改。

我们需要使用pinia提供的defineStore()方法来创建一个store，该store用来存放我们需要全局使用的数据。

首先在项目src目录下新建store文件夹，用来存放我们创建的各种store，然后在该目录下新建user.ts文件，主要用来存放与user相关的store。

代码如下：

```
/src/store/user.ts

import { defineStore } from 'pinia'

// 第一个参数是应用程序中 store 的唯一 id
export const useUsersStore = defineStore('users', {
  // 其它配置项
})
```

创建store很简单，调用pinia中的defineStore函数即可，该函数接收两个参数：

- name：一个字符串，必传项，该store的唯一id。
- options：一个对象，store的配置项，比如配置store内的数据，修改数据的方法等等。

我们可以定义任意数量的store，因为我们其实一个store就是一个函数，这也是pinia的好处之一，让我们的代码扁平化了，这和Vue3的实现思想是一样的。

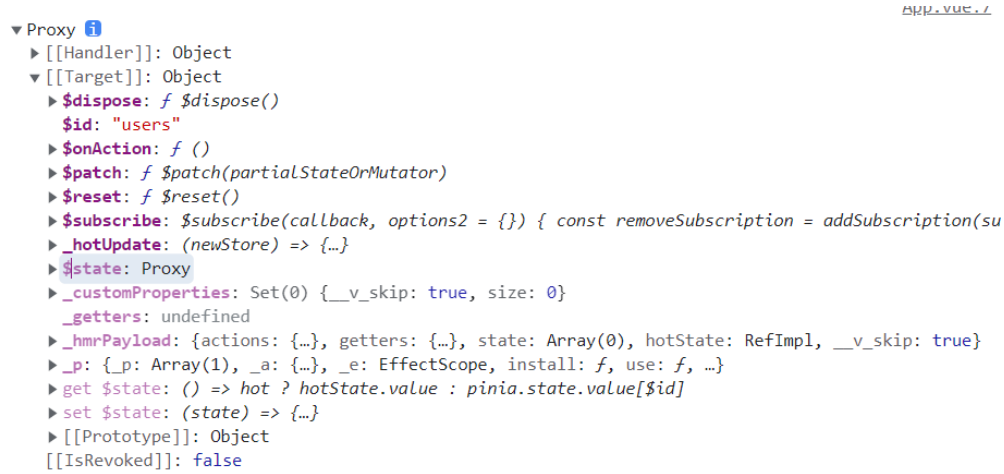
4.3 使用store

前面我们创建了一个store，说白了就是创建了一个方法，那么我们的目的肯定是使用它，假如我们要在App.vue里面使用它，该如何使用呢?

代码如下：

```
/src/App.vue
<script setup lang="ts">
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
console.log(store);
</script>
```

使用store很简单，直接引入我们声明的useUsersStore 方法即可，我们可以先看一下执行该方法输出的是什么：



4.4 添加state

我们都知道store是用来存放公共数据的，那么数据具体存在在哪里呢？前面我们利用defineStore函数创建了一个store，该函数第二个参数是一个options配置项，我们需要存放的数据就放在options对象中的state属性内。

假设我们往store添加一些任务基本数据，修改user.ts代码。

代码如下：

```
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
});
```

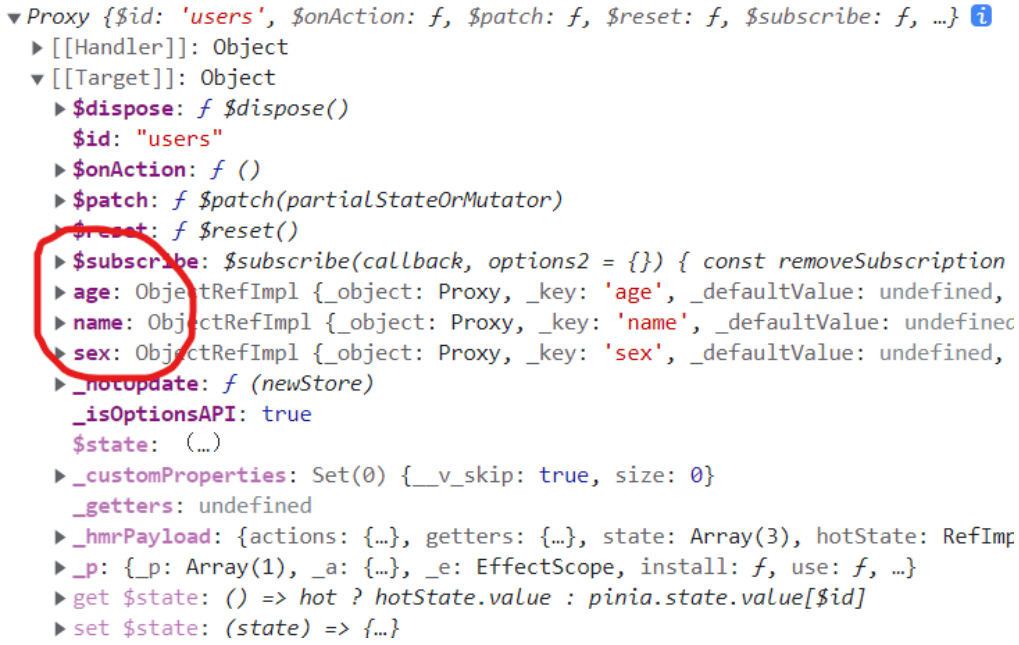
上段代码中我们给配置项添加了state属性，该属性就是用来存储数据的，我们往state中添加了3条数据。需要注意的是，state接收的是一个箭头函数返回的值，它不能直接接收一个对象。

4.5 操作state

我们往store存储数据的目的就是为了操作它，那么我们接下来就尝试操作state中的数据。

4.5.1 读取state数据

读取state数据很简单，前面我们尝试过在App.vue中打印store，那么我们添加数据后再来看看打印结果：



这个时候我们发现打印的结果里面多了几个属性，恰好就是我们添加的数据，修改App.vue，让这几个数据显示出来。

代码如下：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
</template>
<script setup lang="ts">
import { ref } from "vue";
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
const name = ref<string>(store.name);
const age = ref<number>(store.age);
const sex = ref<string>(store.sex);
</script>
```

输出结果：

姓名：小猪课堂

年龄：25

性别：男

上段代码中我们直接通过store.age等方式获取到了store存储的值，但是大家有没有发现，这样比较繁琐，我们其实可以用解构的方式来获取值，使得代码更简洁一点。

解构代码如下：

```
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
const { name, age, sex } = store;
```

上段代码实现的效果与一个一个获取的效果一样，不过代码简洁了很多。

4.5.2 多个组件使用state

我们使用store的最重要的目的就是为了组件之间共享数据，那么接下来我们新建一个child.vue组件，在该组件内部也使用state数据。

child.vue代码如下：

```
<template>
  <h1>我是child组件</h1>
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
</template>
<script setup lang="ts">
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
const { name, age, sex } = store;
</script>
```

child组件和app.vue组件几乎一样，就是很简单的使用了store中的数据。

实现效果：

姓名：小猪课堂

年龄：25

性别：男

我是child组件

姓名：小猪课堂

年龄：25

性别：男

这样我们就实现了多个组件同时使用store中的数据。

4.5.3 修改state数据

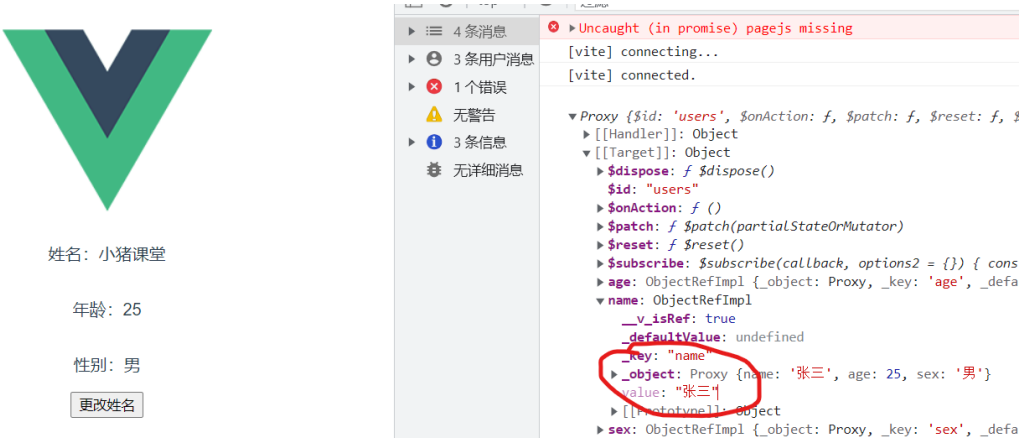
如果我们想要修改store中的数据，可以直接重新赋值即可，我们在App.vue里面添加一个按钮，点击按钮修改store中的某一个数据。

代码如下：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
  <button @click="changeName">更改姓名</button>
```

```
</template>
<script setup lang="ts">
import child from './child.vue';
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
const { name, age, sex } = store;
const changeName = () => {
  store.name = "张三";
  console.log(store);
};
</script>
```

上段代码新增了changeName 方法，改变了store中name的值，我们点击按钮，看看最终效果：



我们可以看到store中的name确实被修改了，但是页面上似乎没有变化，这说明我们的使用的name不是响应式的。

很多小伙伴可能会说那可以用监听函数啊，监听store变化，刷新页面...

其实，pinia提供了方法给我们，让我们获得的name等属性变为响应式的，我们重新修改代码。

app.vue和child.vue代码修改如下：

```
import { storeToRefs } from 'pinia';
const store = useUsersStore();
const { name, age, sex } = storeToRefs(store);
```

我们两个组件中获取state数据的方式都改为上段代码的形式，利用pinia的storeToRefs函数，将sstate中的数据变为了响应式的。

除此之外，我们也给child.vue也加上更改state数据的方法。

child.vue代码如下：

```
<template>
  <h1>我是child组件</h1>
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
  <button @click="changeName">更改姓名</button>
</template>
```

```vue
<script setup lang="ts">
import { useUsersStore } from "../src/store/user";
import { storeToRefs } from 'pinia';
const store = useUsersStore();
const { name, age, sex } = storeToRefs(store);
const changeName = () => {
  store.name = "小猪课堂";
};
</script>
```

这个时候我们再来尝试分别点击两个组件的按钮，实现效果如下：

姓名：张三

年龄：25

性别：男

更改姓名

我是child组件

姓名：张三

年龄：25

性别：男

更改姓名

当我们store中数据发生变化时，页面也更新了！

4.5.4 重置state

有时候我们修改了state数据，想要将它还原，这个时候该怎么做呢？就比如用户填写了一部分表单，突然想重置为最初始的状态。

此时，我们直接调用store的$reset()方法即可，继续使用我们的例子，添加一个重置按钮。

代码如下：

```
<button @click="reset">重置store</button>
// 重置store
const reset = () => {
  store.$reset();
};
```

当我们点击重置按钮时，store中的数据会变为初始状态，页面也会更新。

4.5.5 批量更改state数据

前面我们修改state的数据是都是一条一条修改的，比如store.name=”张三”等等，如果我们一次性需要修改很多条数据的话，有更加简便的方法，使用store的$patch方法，修改app.vue代码，添加一个批量更改数据的方法。

代码如下：

```
<button @click="patchStore">批量修改数据</button>
// 批量修改数据
const patchStore = () => {
  store.$patch({
    name: "张三",
    age: 100,
    sex: "女",
  });
};
```

有经验的小伙伴可能发现了，我们采用这种批量更改的方式似乎代价有一点大，假如我们state中有些字段无需更改，但是按照上段代码的写法，我们必须要将state中的所有字段例举出了。为了解决该问题，pinia提供的$patch方法还可以接收一个回调函数，它的用法有点像我们的数组循环回调函数了。

示例代码如下：

```
store.$patch((state) => {
  state.items.push({ name: 'shoes', quantity: 1 })
  state.hasChanged = true
})
```

上段代码中我们即批量更改了state的数据，又没有将所有的state字段列举出来。

4.5.6 直接替换整个state

pinia提供了方法让我们直接替换整个state对象，使用store的$state方法。

示例代码：

```
store.$state = { counter: 666, name: '张三' }
```

上段代码会将我们提前声明的state替换为新的对象，可能这种场景用得比较少，这里我就不展开说明了。

4.6 getters属性

getters是defineStore参数配置项里面的另一个属性，前面我们讲了state属性。getter属性值是一个对象，该对象里面是各种各样的方法。大家可以把getter想象成Vue中的计算属性，它的作用就是返回一个新的结果，既然它和Vue中的计算属性类似，那么它肯定也是会被缓存的，就和computed一样。

当然我们这里的getter就是处理state数据。

4.6.1 添加getter

我们先来看一下如何定义getter吧，修改user.ts。

代码如下：

```
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
  getters: {
    getAddAge: (state) => {
      return state.age + 100;
    },
  },
});
```

上段代码中我们在配置项参数中添加了getter属性，该属性对象中定义了一个getAddAge方法，该方法会默认接收一个state参数，也就是state对象，然后该方法返回的是一个新的数据。

4.6.2 使用getter

我们在store中定义了getter，那么在组件中如何使用呢？使用起来非常简单，我们修改App.vue。

代码如下：

```
<template>
  <p>新年龄：{{ store.getAddAge }}</p>
```

```
  <button @click="patchStore">批量修改数据</button>
</template>
<script setup lang="ts">
import { useUsersStore } from "../src/store/user";
const store = useUsersStore();
// 批量修改数据
const patchStore = () => {
  store.$patch({
    name: "张三",
    age: 100,
    sex: "女",
  });
};
</script>
```

上段代码中我们直接在标签上使用了store.gettAddAge方法，这样可以保证响应式，其实我们state中的name等属性也可以以此种方式直接在标签上使用，也可以保持响应式。

当我们点击批量修改数据按钮时，页面上的新年龄字段也会跟着变化。

4.6.3 getter中调用其它getter

前面我们的getAddAge方法只是简单的使用了state方法，但是有时候我们需要在这一个getter方法中调用其它getter方法，这个时候如何调用呢?

其实很简单，我们可以直接在getter方法中调用this，this指向的便是store实例，所以理所当然的能够调用到其它getter。

示例代码如下:

```
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
  getters: {
    getAddAge: (state) => {
      return state.age + 100;
    },
    getNameAndAge(): string {
      return this.name + this.getAddAge; // 调用其它getter
    },
  },
});
```

上段代码中我们又定义了一个名为getNameAndAge的getter函数，在函数内部直接使用了this来获取state数据以及调用其它getter函数。

细心的小伙伴可能会发现我们这里没有使用箭头函数的形式，这是因为我们在函数内部使用了this，箭头函数的this指向问题相信大家都知道吧！所以这里我们没有采用箭头函数的形式。

那么在组件中调用的形式没什么变化，代码如下：

```
<p>调用其它getter：{{ store.getNameAndAge }}</p>
```

4.6.4 getter传参

既然getter函数做了一些计算或者处理，那么我们很可能会需要传递参数给getter函数，但是我们前面说getter函数就相当于store的计算属性，和vue的计算属性差不多，那么我们都知道Vue中计算属性是不能直接传递参数的，所以我们这里的getter函数如果要接受参数的话，也是需要做处理的。

示例代码：

```
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
  getters: {
    getAddAge: (state) => {
      return (num: number) => state.age + num;
    },
    getNameAndAge(): string {
      return this.name + this.getAddAge; // 调用其它getter
    },
  },
});
```

上段代码中我们getter函数getAddAge接收了一个参数num，这种写法其实有点闭包的概念在里面了，相当于我们整体返回了一个新的函数，并且将state传入了新的函数。

接下来我们在组件中使用，方式很简单，代码如下：

```
  <p>新年龄：{{ store.getAddAge(1100) }}</p>
```

4.7 actions属性

前面我们提到的state和getters属性都主要是数据层面的，并没有具体的业务逻辑代码，它们两个就和我们组件代码中的data数据和computed计算属性一样。

那么，如果我们有业务代码的话，最好就是卸载actions属性里面，该属性就和我们组件代码中的methods相似，用来放置一些处理业务逻辑的方法。

actions属性值同样是一个对象，该对象里面也是存储的各种各样的方法，包括同步方法和异步方法。

4.7.1 添加actions

我们可以尝试着添加一个actions方法，修改user.ts。

代码如下：

```
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
  getters: {
    getAddAge: (state) => {
      return (num: number) => state.age + num;
    },
    getNameAndAge(): string {
      return this.name + this.getAddAge; // 调用其它getter
    },
  },
  actions: {
    saveName(name: string) {
      this.name = name;
    },
  },
});
```

上段代码中我们定义了一个非常简单的actions方法，在实际场景中，该方法可以是任何逻辑，比如发送请求、存储token等等。大家把actions方法当作一个普通的方法即可，特殊之处在于该方法内部的this指向的是当前store。

4.7.2 使用actions

使用actions中的方法也非常简单，比如我们在App.vue中想要调用该方法。

代码如下：

```
const saveName = () => {
  store.saveName("我是小猪");
};
```

我们点击按钮，直接调用store中的actions方法即可。

5.总结示例代码

前面的章节中的代码都不完整，主要贴的是主要代码部分，我们这节将我们本篇文章用到的所有代码都贴出来，供大家练习。

main.ts代码：

```
import { createApp } from "vue";
import App from "./App.vue";
import { createPinia } from "pinia";
const pinia = createPinia();

const app = createApp(App);
app.use(pinia);
app.mount("#app");
```

user.ts代码：

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

// 第一个参数是应用程序中 store 的唯一 id
export const useUsersStore = defineStore("users", {
  state: () => {
    return {
      name: "小猪课堂",
      age: 25,
      sex: "男",
    };
  },
  getters: {
    getAddAge: (state) => {
      return (num: number) => state.age + num;
    },
    getNameAndAge(): string {
      return this.name + this.getAddAge; // 调用其它getter
    },
  },
  actions: {
    saveName(name: string) {
      this.name = name;
    },
  },
});
```

App.vue代码：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
  <p>新年龄：{{ store.getAddAge(1100) }}</p>
  <p>调用其它getter：{{ store.getNameAndAge }}</p>
  <button @click="changeName">更改姓名</button>
```

```
  <button @click="reset">重置store</button>
  <button @click="patchStore">批量修改数据</button>
  <button @click="saveName">调用aciton</button>

  <!-- 子组件 -->
  <child></child>
</template>
<script setup lang="ts">
import child from "./child.vue";
import { useUsersStore } from "../src/store/user";
import { storeToRefs } from "pinia";
const store = useUsersStore();
const { name, age, sex } = storeToRefs(store);
const changeName = () => {
  store.name = "张三";
  console.log(store);
};
// 重置store
const reset = () => {
  store.$reset();
};
// 批量修改数据
const patchStore = () => {
  store.$patch({
    name: "张三",
    age: 100,
    sex: "女",
  });
};
// 调用actions方法
const saveName = () => {
  store.saveName("我是小猪");
};
</script>
```

child.vue代码：

```
<template>
  <h1>我是child组件</h1>
  <p>姓名：{{ name }}</p>
  <p>年龄：{{ age }}</p>
  <p>性别：{{ sex }}</p>
  <button @click="changeName">更改姓名</button>
</template>
<script setup lang="ts">
import { useUsersStore } from "../src/store/user";
import { storeToRefs } from 'pinia';
const store = useUsersStore();
const { name, age, sex } = storeToRefs(store);
const changeName = () => {
  store.name = "小猪课堂";
```

```
};
</script>
```

总结

pinia的知识点很少，如果你有Vuex基础，那么学起来更是易如反掌。其实我们更应该关注的是它的函数思想，大家有没有发现我们在Vue3中的所有东西似乎都可以用一个函数来表示，pinia也是延续了这种思想。

所以，大家理解这种组合式编程的思想更重要，pinia无非就是以下3个大点：

- state
- getters
- actions

当然，本篇文章只是讲解了基础使用部分，但是在实际工作中也能满足大部分需求了，如果还有兴趣学习pinia的其它特点，比如插件、订阅等等，可以移步官网：pinia官网。