

**生产实习报告**

|  |  |
| --- | --- |
| 姓 名： | 黄力豪 |
| 学 号： | 201720180735 |
| 专 业： | 软件工程 |
| 学 院： | 软件学院 |
| 实习性质： | 集中性实践教学 |
| 实习单位： | 北京得到信息科技有限公司 |
| 实习地点： | 北京 |
| 指导教师： | 简庆 |
| 实习时间： | 2019.7-2019.10 |

实习成绩： 90 指导教师签名： 简庆

东华理工大学教务处

**目录**

[1. 实习目的 3](#_Toc27406495)

[2. 实习概述 3](#_Toc27406496)

[2.1 实习时间 3](#_Toc27406497)

[2.2 实习地点 3](#_Toc27406498)

[3. 实习内容及过程 3](#_Toc27406499)

[3.1 vue3基础 3](#_Toc27406500)

[3.2 vue原理 5](#_Toc27406501)

[4. 实习总结与体会 80](#_Toc27406502)

**生 产 实 习 报 告**

## 实习目的

在得到实习了三个月，主要工作是前端页面开发，同期自学vue3使用及源码以及ts。

具体需要学习内容：

1. Vue3基础，vue3原理，typescript。

## 实习概述

#### 实习时间

2020年7月1日——2020年10月1日

#### 实习地点

实习地点：北京

实习单位：北京得到信息科技有限公司

## 实习内容及过程

### Vue3基础

#### 1.vue3生命周期及nextTick的使用

在vue3中，大部分使用都是先引入后函数调用，学习成本极低，如果有vue2开发经验，应该很容易上手

<script lang="ts">

import { onMounted, onBeforeMount, nextTick} from 'vue'

export default {

name: 'App',

setup() {

nextTick(() => {

console.log('nextTick');

})

onMounted(() => {

console.log('mounted');

})

onBeforeMount(() => {

console.log('beforeMounted');

})

console.log('hello vite Vue3')

}

}

</script>

在vue3中setup（）是入口函数，相当于以前的created 和beforecreated生命周期。 大家可以看到其他生命周期在setup函数中调用即可，我相信这段代码没什么好解释的，输出顺序一看就懂。

#### 2.vue 响应式系统和methods

}<template>

<div>

<h3>vue3响应式系统和methods</h3>

<div>年龄:{{ myAge }}</div>

<div>明年的年龄:{{ mylastAge }}</div>

<button @click="AgeAdd">年龄+1</button>

<div>姓名:{{ myName }}</div>

<div>

爱好:

<div v-for="(hoppy, index) in hoppys" :key="index">{{ hoppy }}</div>

</div>

<div>来自 {{ state1.from }}</div>

</div>

</template>

<script lang="ts">

import HelloWorld from './components/HelloWorld.vue'

import {

ref,

toRefs,

reactive,

watchEffect,

watch,

computed,

onMounted

} from 'vue'

export default {

name: 'App',

setup () {

let myAge = ref(23) //响应式数据

let myName = '黄力豪' //非响应式数据

const state = reactive({

//复杂数据响应式 类似data 基于proxy 操作数组也会触发响应式

hoppys: ['中国象棋', 'javaScript']

})

const state1 = reactive({

// 可以定义多个数据源

from: '江西抚州'

})

watchEffect(() => {

// watch 副作用函数 首次加载会触发,当值发生变化也会触发

console.log('年龄:' + myAge.value)

console.log('爱好:' + state.hoppys)

})

let mylastAge = computed(() => {

return myAge.value + 1

})

setTimeout(() => {

state.hoppys[1] = 'typeScript'

myAge.value += 1

myName = '力豪'

}, 1000)

watch([state.hoppys, myAge], newVal => {

//可以监听多个值

console.log('watch:' + newVal)

})

const methods = {

AgeAdd () {

myAge.value += 1

}

}

return {

myName,

myAge,

...toRefs(state), //将reactive转化为ref

state1,

mylastAge,

...methods

}

}

}

</script>

应该不难看出，在vue3中templeate模板的使用方式基本没有发生任何变化，唯一要注意的就是在模板中使用到的任何响应式数据都要在setup函数中返回（包括方法）。

下面来看ts代码，说是ts，其实目前这些数据都可以使用类型推断出来，其实看起来和js一样。

首先任何在setup函数中返回的数据都可以在模板中使用，只是不是响应式数据而已。

在vue3中笔者目前发现有两种方法可以将非响应式数据转化成响应式数据，那就是ref函数和reactive函数，ref应该是基于Object.defineProperty实现的。而reactive是基于proxy实现的，因此建议普通数据类型使用ref，而复杂数据类型使用proxy。但是要注意对是，使用ref，是操作他的value属性进行改值，但是模板中不需要加上value。

在vue3中computed使用基本和以前一样，只是表达形式改成component API的形式，因此不过多言论。

在vue3中watch 有了一些变化，就是可以同时监听多个值，当其中一项发生变化后则会触发watch。

这里大家可能没有见过的就是watchEffect了，有了解过react的朋友对这个应该很熟悉了。 watchEffct在一开始就会调用一次，当watchEffect中使用到的数据发生变化了就会重新执行一次。

#### 3.props和 ref绑定dom<template>

<div>

<h3 ref="H3">ref,props 和一些小功能</h3>

</div>

</template>

<script>

import { ref, onMounted } from 'vue'

export default {

setup () {

let H3 = ref(null)

onMounted(() => {

H3.value.style.color = 'red'

})

//还有一些小功能

// readonly 数据只读

// shallow -( reactive,ref,readonly) 只代理一层

// toRaw 将reactive或者readonly 的值还原

//markRow 永远不会被代理

//isRef isReactive 等

//unref 如果参数是ref返回他的value否则返回参数

return { H3 }

}

}

</script>

#### 4.组件系统

在vue3中组件有多种定义方式，最大大区别就是在vue3中使用jsx/tsx定义组件比vue2中好用太多了，另外组件涉及到子传父，props，attrs ，emit，slot等因此这次可能会说的比较详细。

<template>

<div>

<h3>组件</h3>

<Hello name="黄力豪" @updateName="updateName"></Hello>

<Age :age="33"></Age>

<myInput>

<template v-slot:desc>

<div>

这是输入框

</div>

</template>

</myInput>

</div>

</template>

<script lang="ts">

import Hello from './TSX/Hello'

import Age from './components/Age.vue'

export default {

components:{

Hello,Age

},

setup(){

const updateName =()=>{

console.log(1);

}

return {updateName}

}

}

</script>

可以看到，我写了hello，age，myInput三个组件，其实也是三种组件。

<template>

<div>

<div>{{ props.age }}</div>

</div>

</template>

<script lang="ts">

import { reactive, isReadonly, toRaw, inject, ref, readonly } from 'vue'

interface Props {

age: number

}

import vuex from '../shared/vuex'

export default {

props: {

age: {

type: Number,

default: 0

}

},

setup (props: Props) {

// props 是readonly

//不要尝试去解构props，因为这样会让props失去响应式

// props.age = 23

// console.log(isReadonly(props))

props = reactive(toRaw(props))

return {

props

}

}

}

</script>

第一种组件和vue2使用基本类似，注意的是setup函数其实有两个入参，第一个是props，值得注意的是不要尝试去解构props，否则会让props失去响应性，除此之外，props是上面提到的readonly，是只读属性，如果尝试修改props会报警告。

下面来看hello组件，hello组件是使用tsx写的

export default {

setup (

props: object,

{ attrs, emit }: { attrs: { name: string }; emit: Function }

) {

return { attrs, emit }

},

render (props: { attrs: { name: string }; emit: Function }) {

return (

<div

onClick={() => {

props.emit('updateName')

}}

>

hello {props.attrs.name}

</div>

)

}

}

tsx语法，直接导出一个ts对象即可，这个对象和使用.vue文件基本类似，只是多了一个render函数用来书写html，另外个人感觉vue3的tsx 比react要好用一些。

首先来看setup函数，这次我给了第二个参数，并对其进行了解构，解构出attrs，和 emit。emit大家都很熟悉，不做多介绍。我们来看attrs，从ts的接口我们不难看出，有一个name属性，没错，就是父组件传过来的name属性。这里有朋友可能就会问了，父传子不是要通过props吗？是的，我们像以前一样使用props也可以，但是使用props必须再上面声明props对象，现在我们可以直接通过attrs取得，何乐而不为。另外注意的是，attrs依旧是可读不可写的，但是不是readonly的，如果你尝试去修改attrs则会报错。如果执行修改，可以尝试根据数据和需要进行深浅拷贝。

下面来看render函数，render函数第一个参数，我们一般也叫props，render的props是一个对象，里面有很多属性，如emit,emit,emit,slots,$attrs等，除此之外，还合并了setup函数等返回值。

attrs和我们的attrs是一样的，因此在这个例子中，我们完全可以使用attrs 和我们的attrs 是一样的，因此在这个例子中，我们完全可以使用attrs和我们的attrs是一样的，因此在这个例子中，我们完全可以使用attrs和$emit代替attrs和emit，而删除setup函数。

我们可以看出，子传父可以使用emit函数，是不是比react的子传父舒服很多？

使用tsx还有一个好处就是，如果使用template模板，我们要求函数必须传number类型的数据，但是在template中传入其他参数类型完全检查不到，而你使用tsx则可以很容易检查出来。

css属性响应式

<template>

<div>

css 属性响应式与指令

<h1 v-highlight="红色">这是一串被高亮为红色的字</h1>

</div>

</template>

<script>

export default {

setup(){

return {

"红色": 'red',

"字体大小": '40px',

}

}

}

</script>

<style vars='{红色, 字体大小}'>

div{

color: var(--红色);

font-size: var(--字体大小);

}

</style>

#### 5.全局通信方式

<template>

<div>

<h3>全局通信</h3>

{{myName}}

<Age></Age>

爱好：{{hoppy}}

</div>

</template>

<script lang="ts">

import {toRefs, provide,ref, inject} from 'vue'

import vuex from './shared/vuex'

import Age from './components/Age.vue'

export default {

components:{

Age

},

setup(){

const {myStore, updateName, updatedAge } = vuex

updateName('力豪')

updatedAge(18)

provide('hoppy',ref('javascript'))

let hoppy = ref(inject('hoppy') as string)

return {...toRefs(myStore),hoppy}

}

}

</script>

provide和inject从vue中导出，在某个组件中使用provide，则其子孙以及子子孙孙都可以拿到provide提供的值。第一个参数是提供的名字，使用字符串或者symbol即可。第二个是传过去的数据， 不过注意要让其成为响应式的话则需要使用ref或者reactive包装一下。子组件使用inject使用即可。 在vue3的vuex中，也是基于此api的

第二种方法

直接创建一个文件，利用es6模块特性，并导出一个对象即可，然后两个组件都引入这个对象即可。个人还是喜欢这种方法的，足够简单清晰而且目前没有发现使用上的bug。

import { reactive } from 'vue';

const myStore = {

myName: '黄力豪',

myAge: 23

};

const updateName = (newName: string) => {

myStore.myName = newName;

};

const updatedAge = (newAge: number) => {

myStore.myAge = newAge;

};

export default { myStore, updateName, updatedAge };

另外这种方式还有一个好处，将导出的对象改成为函数，函数中return这个对象，使用时改为调用函数则这些属性不会共享，则类似vue2中的mixins了。

### Vue3原理

#### 公用方法和接口简介

这些过于简单，只是为了让大家在之后的阅读能了解数据类型，因此这些只写一下代码而不做过多解释 #、# util.ts 写了两个函数判断是否是对象和函数

export const isObject = (target:any) => !!(target && typeof target === 'object')

export const isFunction =(target:any)=>(typeof target ==='function')

import { DefineProperty } from './../interface';

export type strNumSym = string | number | symbol

export type isFunOrObject = Function | undefined | null

export type effectTypeGet = 'get'

export type effectTypeSet = 'set' | 'add'

export type computedOptiobs = DefineProperty | Function

export type \_Function = <T extends object>() =>T

#### reactive简单原理

import { isObject } from "./util"

import { handle } from './basehandles'

export const reactive = <T extends object>(target: T) => {

if (isObject(target)) {

return new Proxy(target, handle)

}

return target

}

这段代码超级简单，只是定义了个泛型函数，其作用就是如过参数是对象，则对其使用Proxy代理，否则直接返回。

}import { strNumSym } from './types/index';

import { reactive } from './reactive'

import { isObject } from './util';

function get<T extends object>(target: T, key: strNumSym, receiver: T) {

let res = Reflect.get(target, key, receiver)

return isObject(target[key]) ? reactive(target[key]) : res;

}

function set<T>(target: any, key: strNumSym, value: T, receiver: object) {

let hasKey = Object.prototype.hasOwnProperty.call(target, key)

let oldval = target[key]

let res = Reflect.set(target, key, value, receiver)

if (!hasKey) {

// do something...

}

else if (value !== oldval) {

// do something....

}

return res

}

export const handle = {

get, set

}

没错，为了简单，我这里都handle只是一个对象，只有get参数和set参数。

get函数相对简单，如果进行取值操作，触发get函数，如果target[key] 是对象则进行深度代理，否则直接返回target[key]。

set函数也比较简单，如果进行了改值操作，则触发set函数。将target[key]改为 新值。

不过改值有添加属性和修改属性值两种，对这两种分别做一个判断，做一些其他操作即可。

值得注意对是，对于数组而言，如果push不但会增加一个属性还会修改数组的length属性。对于这两个分别做了什么操作稍后再分析。

我们先简单分析一下代码，就会发现只有在读值的时候才会进行递归操作使数据变成响应式，而不是一上来就深度递归使所有数据进行响应式。

#### Effcet简单原理。

Effect 有点长，因此打算分三部分写完

首先是第一部分

import { strNumSym, effectTypeSet, effectTypeGet } from './types';

import { Effect, EffectOptions }from './interface'

export const effect = (fn: Function, options:EffectOptions= {

lazy: false

}) => {

let effect = createEffect(fn,options)

if (!options.lazy) {

effect()

}

return effect

}

先看effect函数，接受两个参数，第一个是回调函数，第二个是options函数，关于options有那些类型可以去看一EffectOptions接口。目前只传了一个lazy属性。

如果大家知道watchEffect方法的话，就会知道watchEffect中的回调函数在项目中启动的时候就会立即执行一次。因此，如果options.lazy为false的话，则调用一下回调函数，但是我们还要做一些其他的操作，因此我们来看一下 createEffect 函数

let uid = 0

let activeEffect: Effect

let effectStack: Effect[] = []

function createEffect(fn: Function, options:EffectOptions = {}) {

let effect: Effect = function effectReactive() {

if (!effectStack.includes(effect)) {

try {

effectStack.push(effect)

activeEffect = effect

return fn()

} finally {

effectStack.pop()

activeEffect = effectStack[effectStack.length - 1]

}

}

}

effect.id = uid++

effect.options = options

effect.deps=[]

return effect

}

这段代码uid是一个id标志，activeEffect见名知意，暂时还没有用到，主要是为了将来进行依赖收集，我们暂时不用去管它。最核心的代码就是创建了一个effectStack队列，其主要是为了确认当前活跃Effect。然后执行我们传进入的回调函数，然后删除队列中的最后一个并修改活跃的activeEffect。

好了，我们目前做的仅仅只是让传入的effect回调在执行时就调用函数，它的另外一个功能就是在改值时重新执行一遍函数，那么我们怎么做怎么做？

那就要对其进行依赖收集和触发依赖了

首先看track收集依赖

let targetMap: WeakMap<object, Map<strNumSym, Set<Effect>>> = new WeakMap()

export const track = <T extends object>(target: T, type: effectTypeGet, key: strNumSym) => {

if (activeEffect === undefined) return

let depsMap = targetMap.get(target)

if (!depsMap) {

targetMap.set(target, (depsMap = new Map()))

}

let dep = depsMap.get(key)

if (!dep) {

depsMap.set(key, (dep = new Set()))

}

if (!dep.has(activeEffect)) {

dep.add(activeEffect)

activeEffect.deps.push(dep)

}

}

track函数很简单，无非就是收集依赖到targetMap。我们看一下上面代码执行后targetMap的样子。

targetMap是一个weakMap，键名是target，在上例中即{a:3},值是一个map数据结构。map数据结构的键名是key值，也就是a，值为一个set数据结构，其中每一项是Effect

现在我们可以看到执行完effect后，会根据effect函数中读的属性创建一个收集到的依赖targetMap

那么下面就是触发依赖了

我们看过vue2原理的都知道，在set函数中触发依赖。我们可以看到，我在set函数中写的是do something，现在我们把do something 改成触发依赖

## 实习总结与体会

通过这次实习的学习，了解和学习了很多前端方面的知识。使我对前端学习更加的感兴趣。学习到了最新的vue3使用以及原理。让我养成了在以后的学习中持续学习前端最新技术的能力以及更加热爱前端最新技术。