链滴

Vue3.0 的深入学习

作者：zuimengucheng

原文链接：https://ld246.com/article/1612870787907

来源网站：链滴

Vue3.0来了

2020年09月18日在万众期待中Vue3.0正式版发布于git上并被命名为 ‘One Piece’ 。这也许在这个不平凡的一年里不算一件大事，但是对于我们广大的前端开发者开讲是一个轰动的事情。Vue3.0历经2年多的开发工作，终于带着全新的特性在万众期待中来了。

Vue3.0学习

Vue3.0正式发布之后笔者也和大多数热爱前端的开发者一样迅速的上手了对Vue3.0的采坑之旅(全英当真难啃啊)。

上手准备

安装Vite

vite是伴随这Vue3.0诞生的单文件组件的非打包开发服务器用来进行3.0的编译

安装vite命令

```
//yarn安装
yarn global add create-vite-app
//npm 全局安装
npm i -g create-vite-app
```

创建项目

```
完整命令：create-vite-app 'xxx项目名'
缩写命令：cva 'xxx项目名'
```

Vue3.0中的新特性

Vue3.0虽然进行很大程度的重构但是还是保留了Vue的友好行。Vue3.0可以完美兼容2.0,就是说开发可以用2.0的写法继续在3.0中进行使用并且完美运行。但是并不建议这么做，因为3.0中一些写的语法用起来是真的特别方便使用。

Vue3.0的设计思想

Vue3.0设计模式与2.0不同采用的是多模块架构思想使得项目中引用Vue框架时耦合降低,不必要强依于Vue。比如在项目中使用Vue3.0时可以按需引用使用的功能模块不再整个引用Vue整个的框架,项目包时只会打包相应的功能模块应用体积会大大降低。若需要进行其他平台的适配、基于Vue3.0进行个化的难度也大大降低，只需要重写相应的更新模块即可。

创建应用程序方式改变

Veu3.0中应用程序实例创建的方式也进行了更改，通过使用为动态函数而不是静态函数进行创建应用序实例。这样做好处为：

1. 创建程序实例时可以实现程序实例之间的相互独立，互不干扰。
2. 可以通过摇树算法treeshaking进行打包优化把未使用的模块不进行打包操作减小程序体积。
3. 拥抱微前端

setup()函数

写法如下：

```
setup(porps, context) {
    const formModel = ref({
      name: "zhangsan",
      password: "666",
    });
//需要暴露出去的值再此return出去
    return {
      formModel,
    };
  },
```

此方法可以接收两个参数props和context

props中传递的是一个响应式的值可以通过toRefs进行结构拿到从上层传递下来的值，但是不能直接行结构会导致props中的属性失去响应式。

context中传递的是上下文在这里可以找到 attrs 和 slots。

此方法是Vue3.0新特性中特别重要的一个特性它的诞生解决了2.0中许多遗留的问题。

1. setup()的出现使得代码的可维护性、复用性、可读性都大大提升了
2. setup()的出现可以替代掉mixin()函数，解决掉了mixin()可能存在命名重复的隐患
3. 消除this，拥抱ts
4. setup中的数据优先级高与data()中Vue3.0中获取某个值首先会从setupDate中进行查找。

5. 更小的性能消耗，在2.0中data中所用的定义的数据都会进行响应式处理不考虑使用者是否需要使相应式导致性能消耗较高。在3.0中是通过ref()和reactive()进行按需响应式处理。不仅把选择权交给开发者，还大大降低了性能消耗。

v-model的改变

总所周知在Vue2.0中v-model只能进行动态绑定value值的变化，但是在Vue3.0中v-mode可以同时定多个不同值的写法为v-model:xxx='变量名'，子组件需要更改绑定的值时需要使用setup函数中的二个参数进行触发更改方法，写法为context.emit('update:xxx', false)

示例：

```
<zuiMengDialog v-model:visible="x" :closeOnClickOverlay="false" :save="f1" :cancel="f2"  >
       <template v-slot:content>
          <strong>hi word</strong>
          <div>你好啊世界</div>
       </template>
       <template v-slot:title>
         这个是一个弹出框
       </template>
        <template v-slot:footer>
        <zuiMengButton theme="button" level="main" @click="toggle">确认</zuiMengButto
>
        <zuiMengButton @click="toggle" >取消</zuiMengButton>
       </template>
   </zuiMengDialog>
//子组件中更改父组件绑定值方法
<script lang="ts" setup='props,context' >
import { SetupContext } from 'vue';
import {zuiMengButton} from './index'
    export const close = () => {
          //visible为需要更改的值名字
          context.emit('update:visible', false)
       }

</script>
```

ref()和reactive()

这两个方法都是Vue3.0中暴露出的进行响应式绑定的函数。但是两者不同点在于：

ref(null)一般处理是基础数据类型生成的响应式对象获取值或者更改值需要使用value才能获取到对应值，不仅如此Vue3.0取消了this.$refs进行组件的绑定，而是通过ref()绑定组件实例进行相应操作。

reactive(null)一般处理的是复杂数据类型，生成的响应式对象获取值则可以直接使用和修改。

```
setup(props, context) {
  let count=ref(1)
  console.log(count.value)//1
  count.value=2
  console.log(count.value)//2
  let state = reactive({
    stateSon: 1,
```

```
    stateDaughter: 2
  })
    console.log(state .stateSon)//1
    state.stateSon=2
    console.log(state .stateSon)//2

  return {
     count,
     state
  }
}
```

template模板部分

Vue3.0中由于通过Fragment()函数巧妙进行切片处理，在template中可以进行多个根节点的书写。2.0中是无法进行实现，无疑是一个重大的突破。

watchEffect()函数

此函数是一个监听函数，在props中数据第一次初始化和更改时进行触发，类似于React中的useEffect)函数。需要留意的是这个方法在初始化时会在onMounted周期之前调用一次，但是此时dom节点未载，若进行操作dom会报错。

此方法传递两个参数

第一个参数为回调函数(effect()副作用函数)，effect()中还能接收一个参数onInvalidate()清除副作用数，主要作用就是在组件销毁是进行 effect()的清除操作。这个思想和react中很相似，当你使用监听后需要在组件销毁是进行清除监听。

第二个参数为 options他的作用是一个指针调度器用来控制watchEffect的触发事件是在数据的更改前还是在数据的更改之后flush有两个值'post'(数据更新之后调用)和'pre'(数据更新之前调用)

```
setup(props, context) {
  watchEffect(() => {
            //回调函数主体
          }, {
            //options值
             flush: 'post'
          })
       })
}
```

watch() 监听器

watch作用和之前的2.0中并没有两样但是书写方式却进行也改变

在setup函数中watch监听单个的写法

```
setup(props, context) {
  let count=ref(1)
  let state = reactive({
     stateSon: 1
  })
```

```
    //监听reactive类型
    watch(
    () => state.stateSon，  //需要被监听的数据
    (newVlaue，  oldVlaue) => { //参数1为新值，  参数2为旧值
        console.log(newVlaue) //数据变化后执行的回调函数
    },
    {lazy：  false}//首次创建是否监听
)

    //监听ref类型
     watch(count1，(newVlaue，  oldVlaue) => {//参数1为新值，  参数2为旧值
        console.log(newVlaue) //数据变化后执行的回调函数
     })

    return {
        count,
        state
    }
}
```

监听多个数据写法

```
setup(props, context) {
    let count=ref(1)
    let countTwo=ref(2)

    let state = reactive({
        stateSon: 1,
        stateDaughter: 2
    })
    //监听reactive类型
    watch(
[ () => state.stateSon, () => state.stateDaughter]//需要被监听的数据
    ,
    ([sonNewVlaue,aughterNewVlaue]，[sonOldVlaue,aughterOldVlaue]) => { //参数1为新值，
数2为旧值
        console.log('回调函数触发了') //数据变化后执行的回调函数
    },
    {lazy：  false}//首次创建是否监听
)

    //监听ref类型
     watch([count,countTwo]，([countNewVlaue,countTwoNewVlaue]，[countOldVlaue,countT
oOldVlaue]) => {//参数1为新值，  参数2为旧值
        console.log('回调函数触发了') //数据变化后执行的回调函数
     })

    return {
        count,
        countTwo,
        state
    }
}
```

生命周期变化

在Vue3.0中生命周期也是按照2.0写法进行调用，在基础上又进行了优化可以按需进行导入在setup()数中进行调用

但是需要注意的是created()和beforeCreated()生命周期无法在setup()函数中进行使用他们执行之间setup()函数之前

```
import { onBeforeMount, onMounted, updated } from '@vue/composition-api'
setup () {
  onBeforeMount(() => {
    console.log('onBeforeMount生命周期被调用了')
  })
  onMounted(() => {
    console.log('onMounted生命周期被调用了')
  })
  updated(() => {
    console.log('updated生命周期被调用了')
  })
}
```

响应式数据实现的改变

Vue2.0是通过Object.defineProperty进行数据劫持给每个数据加上set和get方法同时在数据的get()法触发时进行更新函数依赖的收集，在数据的set()触发时循环遍历调用收集起来的更新函数进行视图更新来实现数据响应式

Vue3.0中通过实现Proxy(代理)和Reflect(映射)来实现数据响应式，完美解决Vue2.0响应式一些弊端如对象或者数组新增属性无法实现响应式需要手动进行添加、数组实现响应式需要额外进行处理、响式数据过多性能问题(因为Vue3.0中实现响应式时进行了懒执行优化不管数据有多大只要用户不访问就不会进行响应式处理)

computed()计算属性的变化

vue2.0中计算属性是只读的不能进行更改，但是在3.0中computed可以进行值的修改

```
setup () {
     const count = ref(1)
     const computedDate= computed(() => count.value + 1) //不允许修改返回count的值+1

     const computedDateTwo= computed({
        get:() =>count.value + 666,
        set: (value) => count.value = value
     })
     // addCount2.value = 123   //赋值方法

    return {
       count,
       computedDate,
       computedDateTwo
    }
  }
```

Vue3.0中移除的部分属性

this.$refs.xxx Vue3.0移除了通过此方法进行组件实例的绑定但是新的写法就是通过ref(null)进行绑定法如下

```
<template>
  <demo> 常规使用 </demo>
  <div>
    <zuiMengForm :model="formModel" :rules="rules" ref="ceShiForm">
      <zuiMengFormItem lable="密码" @click="clickCeShi" prop="password">
        {{ formModel.password }}
      </zuiMengFormItem>
    </zuiMengForm>
  </div>
</template>

setup(){
const ceShiForm = ref(null);
const clickCeShi = () => {
      ceShiForm.value.validate();
    };

}
```

this.$emit

this.$on这两个属性方法在Vue中被移除了Vue3.0中认为这个不是vue应该做的事情就直接移除了，要在继续使用这两个方法需要进行第三方差劲进行使用

Vue3.0中的优化策略

静态节点提升

vue3.0进行编译时会将节点进行分类存储如果需要编译的是静态节点，就将节点存储到编译函数外，续更新时直接使用，不需要再进行编译更新，大大节省了编译时间

补丁标记和动态属性记录

Vue3.0中进行模板编译时遇到节点中需要动态编译元素，会根据元素类别不同，通过位运算进行标记标记之后diff算法渲染时根据标记的不同进行不同的处理，减少部分不必要操作，大大提高的渲染性。

缓存事件处理程序

编译器在编译回调函数时，会优先从缓存中进行查找，若找到则进行使用，查找不到则进行生成新的的回调函数，同时存放到缓存中。方便下次更新时使用，避免每次重新创建大大减少了性能损耗

区块block

对于一个父元素存在很多个子元素，编译器进行出处理时会将此父元素下所用的动态子元素查找出来同时存放到以此父元素创建的block对象的动态子元素数组中。后面进行更新时diff算法会直接比较动

子元素数组中元素变化进行更新大大提升更新效率。

总结

Vue在2.0的基础上进行了翻天覆地式的改变不仅将2.0中存在的缺点进行了优化，还完全向下兼容对前版本的Vue的开发者特别友好，再加上Vue3.0中新的优化策略对资源的节约和性能的提升使Vue3.更加强大更值得去使用开发项目，虽然美中不足的兼容性暂时未进行优化(Vue3.0基本上已经放弃了IE但是不妨碍Vue3.0雄起，期待Vue3.0在以后大放异彩。

附：自己学习Vue3.0中的练手项目zuimengUI

此次分享到此结束欢迎大家来相互学习交流