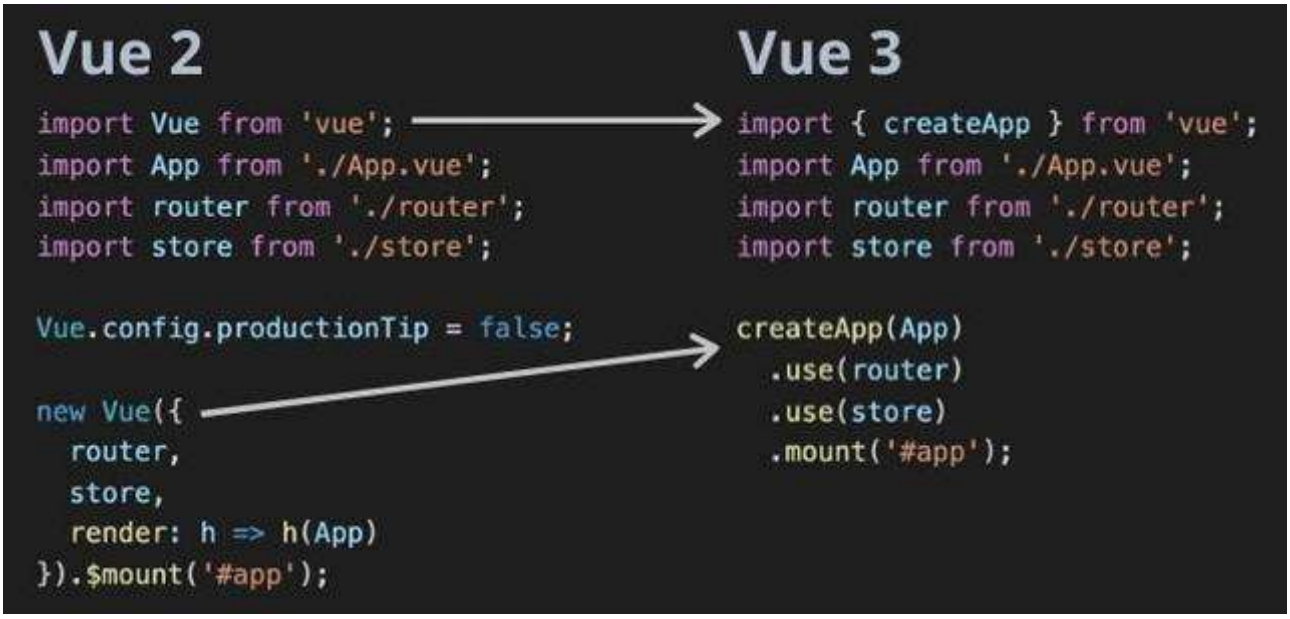
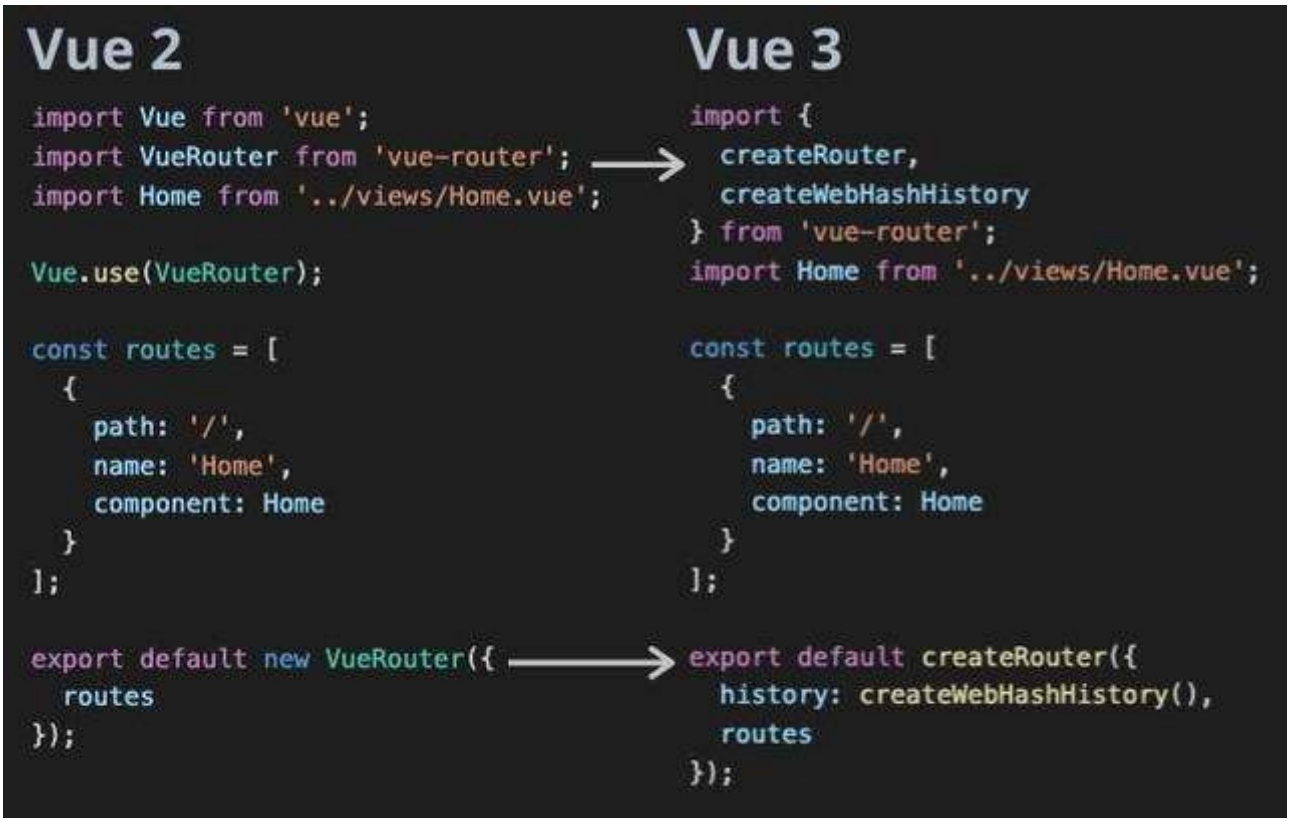
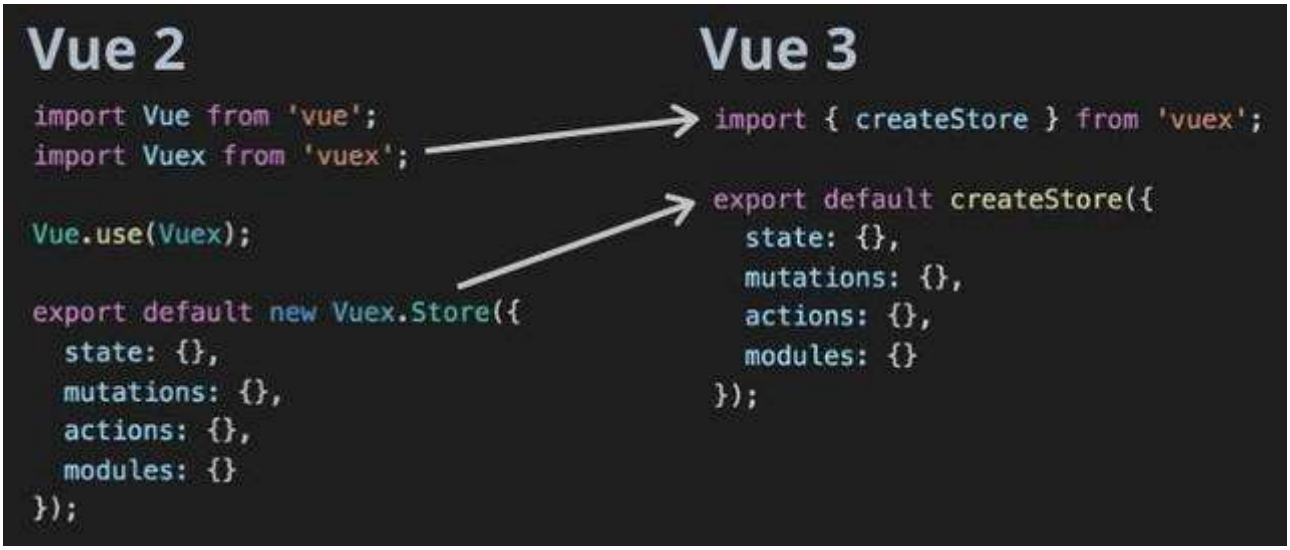
1-淡定围观


Vue 2
import Vue from 'vue';
import App from './App.vue';
import router from './router';
import store from './store';
Vue.config.productionTip = false;
new Vue({
router,
store,
render: h => h(App)
}).$mount('#app');
Vue 3
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';
import store from './store';
createApp(App)
.use(router)
.use(store)
.mount('#app');


Vue 2
import Vue from 'vue';
import VueRouter from 'vue-router';
import Home from '../views/Home.vue';
Vue.use(VueRouter);
const routes = [
{
path: '/',
name: 'Home',
component: Home
}
];
export default new VueRouter({
routes
});
Vue 3
import {
createRouter,
createWebHashHistory
} from 'vue-router';
import Home from '../views/Home.vue';
const routes = [
{
path: '/',
name: 'Home',
component: Home
}
];
export default createRouter({
history: createWebHashHistory(),
routes
});



> 报错Resolve error: Cannot find module 'vue-loader-v16/package.json' - **更新node到最新版**

> 热更新有问题， 需要及时**手动刷新页面**

2-介绍

Vue3.0设计目标

- 更小
  - 全局 API 和内置组件 / 功能支持 tree-shaking
  - 常驻的代码尺寸控制在 10kb gzipped 上下
- 更快
  - 基于 Proxy 的变动侦测，性能整体优于 getter / setter
  - Virtual DOM 重构
  - 编译器架构重构，更多的编译时优化
- 加强API设计一致性
- 加强TypeScript支持
- 提高自身可维护性
  - 代码采用 monorepo 结构，内部分层更清晰
  - TypeScript 使得外部贡献者更有信心做改动
- 开放更多底层功能

3-Composition API介绍

起初定义的是Vue-Function-API，后经过社区意见收集，更名为Vue-Composition-API.

3-1reactive

作用：创建响应式对象，非包装对象，可以认为是模板中的状态。

- template 可以放兄弟节点
- reactive 类似useState, **如果参数是字符串，数字,会报警告，value cannot be made reactive**, 所以应该设置对象，这样可以数据驱动页面

```
<div>
  {{countobj.count}}-<button @click="add">add</button>
</div>

setup () {

  const countobj = reactive({
    count: 0
  })
  const add = () => {
    countobj.count++
  }
  return {
    countobj,
    add
  }
}
```

3-2ref

作用：创建一个包装式对象，含有一个响应式属性value。 它和reactive的差别，就是前者没有包装属性value

const count = ref(0) ，可以接收普通数据类型,count.value++

```
<div>
  {{count}}-<button @click="add">add</button>
</div>

setup () {
  const add = () => {
    count.value++
  }
  const count = ref(0)

  return {
    count,
    add
  }
}
```

3-2-1ref嵌套在reactive中

```
<template>
  <div class="home">
    home-{{count}}--{{state.count}}
    <button @click="add">click</button>
  </div>
</template>

<script>
import { reactive, ref } from 'vue'
export default {
  name: 'Home',
  setup () {
    const count = ref(0)
    const state = reactive({
      count
    })
    const add = () => {
      state.count++
      //state.count 跟ref count 都会更新
    }
    return {
      state,
      add,
      count
    }
  }
}
</script>
```

3-2-2toRefs

默认直接展开state，那么此时reactive数据变成普通数据，通过toRefs，可以把reactive里的每个属性，转化为ref对象，这样展开后，就会变成多个ref对象， 依然具有响应式特性

```
<template>
  <div class="home">
    home-{{count}}
    <button @click="add">click</button>
  </div>
</template>

<script>
import { reactive, toRefs } from 'vue'
export default {
  name: 'Home',
  setup () {
    const state = reactive({
      count: 1
    })

    const add = () => {
      state.count++
    }
    return {
      ...toRefs(state),
      add
    }
  }
}
</script>
```

3-2-3 ref访问dom或者组件

```
<input type="text" ref="myinput"/>

//js
const myinput = ref(null)

console.log(myinput.value.value)
```

3-3 prop & emit

```
props:["mytite"], //正常接收
setup (props, { emit }) {
    console.log(props.mytitle)
    const handleClick = () => {
      emit('kerwinevent')
    }

    return {
      handleClick
    }
  }
```

3-4生命周期

原方法	升级后
beforeCreate	setup
created	setup
beforeMount	onBeforeMount
mounted	onMounted
beforeUpdate	onBeforeUpdate
updated	onUpdated
beforeDestroy	onBeforeUnmount
destroyed	onUnmounted

原方法beforeDestroy=>原方法beforeUnmount

原方法destroyed=>原方法unmounted

```
import { onUnmounted, onMounted } from 'vue'

setup () {
    ...
    onMounted(() => {
      console.log('onMounted')
    })
    ...
  }
```

3-5计算属性

computed(回调函数)

```
  setup () {
    const mytext = ref('')

    const computedSum = computed(() => mytext.value.substring(0, 1).toUpperCase() +
mytext.value.substring(1))
    // 注意mytext.value
    return {
      mytext,
      computedSum
    }
  }
```

3-6watch

监听器 watch 是一个方法，它包含 2 个参数

```
const reactivedata = reactive({count:1})
const text= ref("")
watch(() => reactivedata.count,
  val => {
    console.log(`count is ${val}`)
  })

watch(text,
  val => {
    console.log(`count is ${val}`)
  })
```

第一个参数是监听的值，count.value 表示当 count.value 发生变化就会触发监听器的回调函数，即第二个参数，第二个参数可以执行监听时候的回调

3-7自定义hooks

虽然composition api 比之前写法好像更麻烦了，但是用上自定义hooks就可以实现函数编程的复用了，更加简洁高效了。一直在模仿。

```
import {
    ref
} from 'vue';

function useCount(){
    const count = ref(1);
    const addCount = (num = 1) => count.value += num;
    return {
        count,
        addCount
    }
}

export {useCount}
```

4路由

4-1配置

```
import { createRouter, createWebHashHistory } from 'vue-router'

const router = createRouter({
  history: createWebHashHistory(''), //hash模式
  routes
})

const router = createRouter({
  history: createWebHistory('/v5'), //history模式
  routes
})

//重定向


  {
    path: '/:kerwin',
    redirect: {
      name: 'film' //命名路由写法
    }
  }
```

4-2获取$router

```
import { getCurrentInstance } from 'vue'

setup(){
    // const router =  useRouter() vue-router中的useRouter直接获取router对象

    const { ctx } = getCurrentInstance() //必须setup中定义
    // ctx.$router == this.$router(之前写法)

    //编程式导航
    ctx.$router.push('/about')
}



//获取动态路由参数
  setup () {
    const router = useRouter()
    console.log(router.currentRoute.value.params.id)
  }

  // 第二种方案
  import { useRoute } from 'vue-router'
  const route = useRoute() //被proxy拦截代理的proxy对象，  可以直接访问属性
  console.log(route.params.id)
```

5 vuex

入口配置createApp(App).use(router).use(store).mount('#app')

```
    {{storeCount}}

    setup () {
      // const store =  useStore() vuex中的useStore直接获取store对象
      // store.commit
      // store.dispath
      // store.state

      const { ctx } = getCurrentInstance()
      const storeCount = computed(() => ctx.$store.state.count)

      add(){
          ctx.$store.commit("addMutation")
      }

      return {
        storeCount
      }
    }
  }
```

```
// store/index.js
export default Vuex.createStore({
  state: {
    count: 1
  },
  mutations: {
    addMutation (state) {
      state.count++
    }
  },
  actions: {
  },
  modules: {
  }
})
```

> 不能使用mapMutations ,mapState....，因为依赖于this.$store

5-1 vuex替代方案

> provide、inject 是 vue-composition-api 的一个新功能：依赖注入功能

```
import { provide, inject } from 'vue'

// 根组件 共享自己的状态
    const kerwinshow = ref(true)
    provide('kerwinshow', kerwinshow)

// detail组件
    onMounted(() => {

      const kerwinshow = inject('kerwinshow')
      kerwinshow.value = false
    })
```