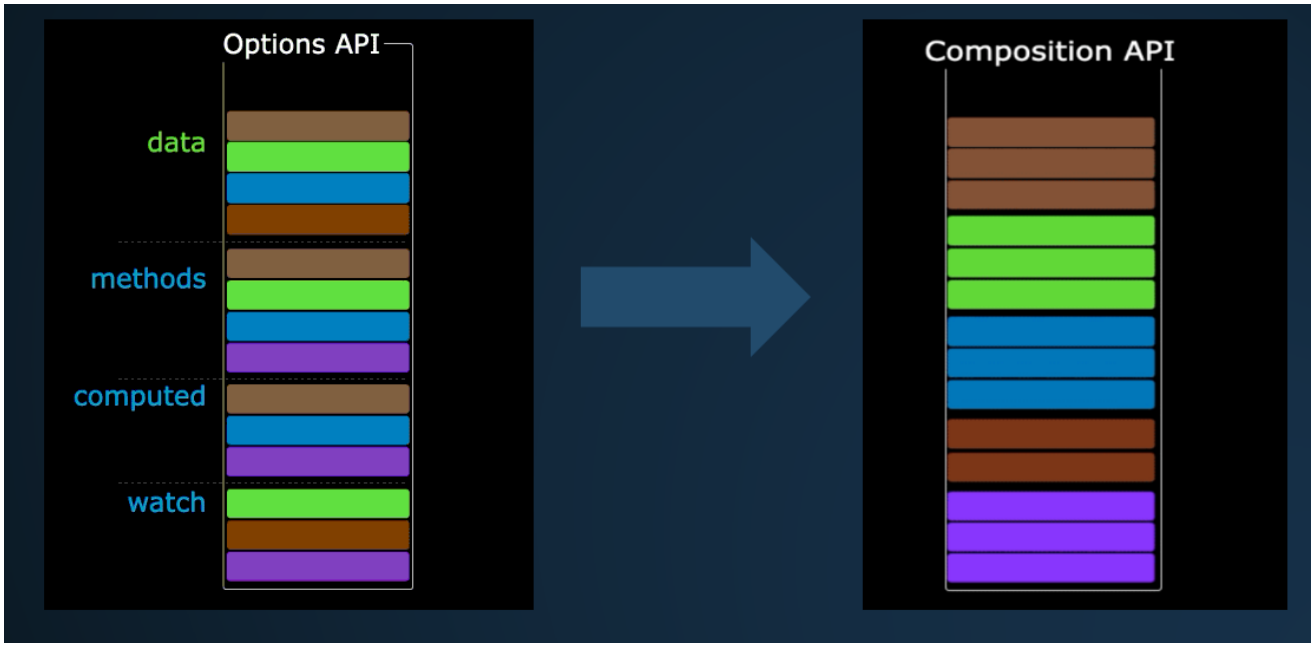
四.vue3组合式API

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



1. reactive

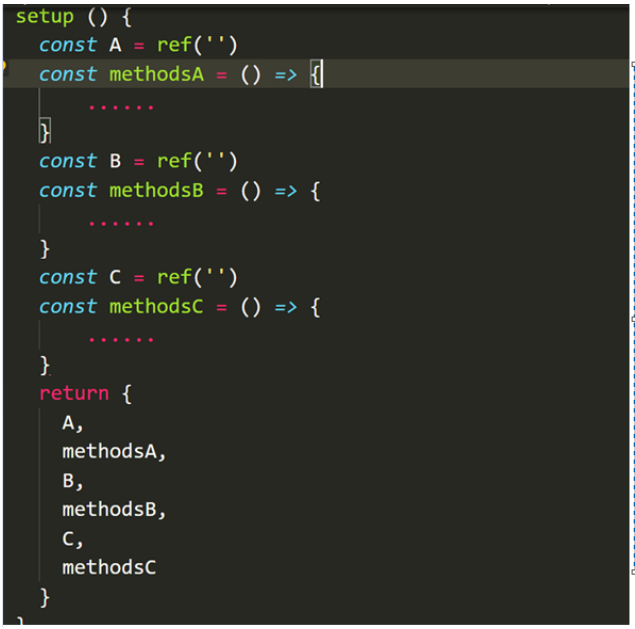
作用：创建响应式对象，非包装对象，可以认为是模板中的状态。

- 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
  }
 }
```



```
setup () {
  const A = ref('')
  const methodsA = () => {
      ......
  }
  const B = ref('')
  const methodsB = () => {
      ......
  }
  const C = ref('')
  const methodsC = () => {
      ......
  }
  return {
    A,
    methodsA,
    B,
    methodsB,
    C,
    methodsC
  }
}
```

```
import useA from './moduleA'
import useB from './moduleB'
import useC from './moduleC'
export default {
  setup () {
    const { A, methodsA } = useA()
    const { B, methodsB } = useB()
    const { C, methodsC } = useC()
    return {
      A,
      methodsA,
      B,
      methodsB,
      C,
      methodsC
    }
  }
}
```

2. ref

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
  }
}
```

2-1 ref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>
```

2-2 toRefs

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>
```

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. 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
  }
 }
```

4. watch

计算属性允许我们声明性地计算衍生值。然而在有些情况下，我们需要在状态变化时执行一些“副作用”：例如更改 DOM，或是根据异步操作的结果去修改另一处的状态。

在组合式 API 中，我们可以使用 `watch` 函数在每次响应式状态发生变化时触发回调函数：

```
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}`)
 })

watch(source, (newValue, oldValue) => {
 // 立即执行，且当 `source` 改变时再次执行
 }, { immediate: true })
```

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

5. VCA中的watchEffect函数

注意：

（1）watch：

- 具有一定的惰性lazy 第一次页面展示的时候不会执行，只有数据变化的时候才会执行
- 参数可以拿到当前值和原始值
- 可以侦听多个数据的变化，用一个侦听器承载

```
const todoId = ref(1)
const data = ref(null)

watch(todoId, async () => {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
  )
  data.value = await response.json()
}, { immediate: true })
```

（2）watchEffect：

- 立即执行，没有惰性，页面的首次加载就会执行。
- 自动检测内部代码，代码中有依赖 便会执行
- 不需要传递要侦听的内容 会自动感知代码依赖，不需要传递很多参数，只要传递一个回调函数
- 不能获取之前数据的值 只能获取当前值
- 一些异步的操作放在这里会更加合适

```
watchEffect(async () => {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
  )
  data.value = await response.json()
})
```

6. prop & emit

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

   return {
    handleClick
   }
 }
```

7. VCA中provide&inject

> provide、inject 是 vue-composition-api 的一个功能：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
   })
```

8. VCA中的生命周期

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

setup () {
   ...

 onMounted(() => {

 console.log('onMounted')
   })
   ...
 }
```

9. 在单文件组件使用VCA的语法糖