Composition API

官方文档: https://v3.cn.vuejs.org/guide/composition-api-introduction.html

拉开序幕的setup

理解：Vue3.0中一个新的配置项，值为一个函数。

setup是所有Composition API（组合API）“ 表演的舞台 ” 。

组件中所用到的：数据、方法等等，均要配置在setup中。

setup函数的两种返回值：

若返回一个对象，则对象中的属性、方法, 在模板中均可以直接使用。（重点关注！）

若返回一个渲染函数：则可以自定义渲染内容。（了解）

注意点：

尽量不要与Vue2.x配置混用

Vue2.x配置（data、methos、computed...）中可以访问到setup中的属性、方法。

但在setup中不能访问到Vue2.x配置（data、methos、computed...）。

如果有重名, setup优先。

setup不能是一个async函数，因为返回值不再是return的对象, 而是promise, 模板看不到return对象中的属性。（后期也可以返回一个Promise实例，但需要Suspense和异步组件的配合）

ref函数

作用: 定义一个响应式的数据

语法: const xxx = ref(initValue)

创建一个包含响应式数据的引用对象（reference对象，简称ref对象）。

JS中操作数据： xxx.value

模板中读取数据: 不需要.value，直接： {{xxx}}

备注：

接收的数据可以是：基本类型、也可以是对象类型。

基本类型的数据：响应式依然是靠Object.defineProperty()的get与set完成的。

对象类型的数据：内部 “ 求助 ” 了Vue3.0中的一个新函数—— reactive函数。

reactive函数

作用: 定义一个对象类型的响应式数据（基本类型不要用它，要用ref函数）

语法：const 代理对象= reactive(源对象)接收一个对象（或数组），返回一个代理对象（Proxy的实例对象，简称proxy对象）

reactive定义的响应式数据是“深层次的”。

内部基于 ES6 的 Proxy 实现，通过代理对象操作源对象内部数据进行操作。

Vue3.0的响应式

实现原理:

通过Proxy（代理）: 拦截对象中任意属性的变化, 包括：属性值的读写、属性的添加、属性的删除等。

通过Reflect（反射）: 对源对象的属性进行操作。

MDN文档中描述的Proxy与Reflect：

Proxy：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Proxy

Reflect：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Reflect

```
new Proxy(data, {
        // 拦截读取属性值
    get (target, prop) {
        return Reflect.get(target, prop)
    },
    // 拦截设置属性值或添加新属性
    set (target, prop, value) {
        return Reflect.set(target, prop, value)
    },
    // 拦截删除属性
    deleteProperty (target, prop) {
        return Reflect.deleteProperty(target, prop)
    }
})

```

reactive对比ref

从定义数据角度对比：

ref用来定义：基本类型数据。

reactive用来定义：对象（或数组）类型数据。

备注：ref也可以用来定义对象（或数组）类型数据, 它内部会自动通过reactive转为代理对象。

从原理角度对比：

ref通过Object.defineProperty()的get与set来实现响应式（数据劫持）。
reactive通过使用Proxy来实现响应式（数据劫持），并通过Reflect操作源对象内部的数据。
从使用角度对比：
ref定义的数据：操作数据需要.value，读取数据时模板中直接读取不需要.value。
reactive定义的数据：操作数据与读取数据：均不需要.value

setup的两个注意点

setup执行的时机
在beforeCreate之前执行一次，this是undefined。
setup的参数
props：值为对象，包含：组件外部传递过来，且组件内部声明接收了的属性。
context：上下文对象
attrs: 值为对象，包含：组件外部传递过来，但没有在props配置中声明的属性, 相当于 this.$attrs。
slots: 收到的插槽内容, 相当于 this.$slots。
emit: 分发自定义事件的函数, 相当于 this.$emit

watch函数

与Vue2.x中watch配置功能一致
两个小“坑”：
监视reactive定义的响应式数据时：oldValue无法正确获取、强制开启了深度监视（deep配置失效）。
监视reactive定义的响应式数据中某个属性时：deep配置有效。

watchEffect函数

watch的套路是：既要指明监视的属性，也要指明监视的回调。
watchEffect的套路是：不用指明监视哪个属性，监视的回调中用到哪个属性，那就监视哪个属性。
watchEffect有点像computed：
但computed注重的计算出来的值（回调函数的返回值），所以必须要写返回值。
而watchEffect更注重的是过程（回调函数的函数体），所以不用写返回值。

自定义hook函数

什么是hook？—— 本质是一个函数，把setup函数中使用的Composition API进行了封装。

类似于vue2.x中的mixin。

自定义hook的优势: 复用代码, 让setup中的逻辑更清楚易懂

shallowReactive 与 shallowRef

shallowReactive：只处理对象最外层属性的响应式（浅响应式）。

shallowRef：只处理基本数据类型的响应式, 不进行对象的响应式处理。

什么时候使用?

如果有一个对象数据，结构比较深, 但变化时只是外层属性变化 ===> shallowReactive。

如果有一个对象数据，后续功能不会修改该对象中的属性，而是生新的对象来替换 ===> shallowRef

readonly 与 shallowReadonly

readonly: 让一个响应式数据变为只读的（深只读）。

shallowReadonly：让一个响应式数据变为只读的（浅只读）。

应用场景: 不希望数据被修改时。

toRaw 与 markRaw

toRaw：

作用：将一个由reactive生成的响应式对象转为普通对象。

使用场景：用于读取响应式对象对应的普通对象，对这个普通对象的所有操作，不会引起页面更新。

markRaw：

作用：标记一个对象，使其永远不会再成为响应式对象。

应用场景:

有些值不应被设置为响应式的，例如复杂的第三方类库等。

当渲染具有不可变数据源的大列表时，跳过响应式转换可以提高性能

provide 与 inject

作用：实现**祖与后代组件间**通信

套路：父组件有一个`provide`选项来提供数据，后代组件有一个`inject`选项来开始使用这些数据

具体写法

祖组件中：

```
setup(){
        ......
    let car = reactive({name:'奔驰',price:'40万'})
    provide('car',car)
    ......
}
```

后代组件中：

```
setup(props,context){
        ......
    const car = inject('car')
    return {car}
        ......
}
```

响应式数据的判断

isRef: 检查一个值是否为一个 ref 对象

isReactive: 检查一个对象是否是由 reactive 创建的响应式代理

isReadonly: 检查一个对象是否是由 readonly 创建的只读代理

isProxy: 检查一个对象是否是由 reactive 或者 readonly 方法创建的代理

Composition API 的优势

我们可以更加优雅的组织我们的代码，函数。让相关功能的代码更加有序的组织在一起。

新的组件

Fragment

在Vue2中: 组件必须有一个根标签

在Vue3中: 组件可以没有根标签, 内部会将多个标签包含在一个Fragment虚拟元素中

好处: 减少标签层级, 减小内存占用

Teleport

什么是Teleport？—— Teleport 是一种能够将我们的**组件html结构**移动到指定位置的技术。

Suspense

等待异步组件时渲染一些额外内容，让应用有更好的用户体验

使用步骤：

异步引入组件

```
import {defineAsyncComponent} from 'vue'
const Child = defineAsyncComponent(()=>import('./components/Child.vue'))
```

使用Suspense包裹组件，并配置好default与 fallback

```
<template>
        <div class="app">
                <h3>我是App组件</h3>
                <Suspense>
                        <template v-slot:default>
                                <Child/>
                        </template>
                        <template v-slot:fallback>
                                <h3>加载中.....</h3>
                        </template>
                </Suspense>
        </div>
</template>

```

其他

全局API的转移

Vue 2.x 有许多全局 API 和配置。

例如：注册全局组件、注册全局指令等

```
//注册全局组件
```

```
Vue.component('MyButton', {
  data: () => ({
    count: 0
  }),
  template: '<button @click="count++">Clicked {{ count }} times.</button>'
})

//注册全局指令
Vue.directive('focus', {
  inserted: el => el.focus()
}
```

Vue3.0中对这些API做出了调整

将全局的API，即：Vue.xxx调整到应用实例（app）上

2.x 全局 API（Vue） 3.x 实例 API (app)

Vue.config.xxxx app.config.xxxx

Vue.config.productionTip 移除

Vue.component app.component

Vue.directive app.directive

Vue.mixin app.mixin

Vue.use app.use

Vue.prototype app.config.globalPropertie

其他改变

data选项应始终被声明为一个函数。

过度类名的更改：

Vue2.x写法

```
.v-enter,
.v-leave-to {
  opacity: 0;
}
.v-leave,
.v-enter-to {
  opacity: 1;
}
```

Vue3.x写法

```
.v-enter-from,
.v-leave-to {
  opacity: 0;
}

.v-leave-from,
.v-enter-to {
  opacity: 1;
}
```

移除keyCode作为 v-on 的修饰符，同时也不再支持`config.keyCodes`

移除`v-on.native`修饰符

父组件中绑定事件

```
<my-component
  v-on:close="handleComponentEvent"
  v-on:click="handleNativeClickEvent"
/>
```

子组件中声明自定义事件

```
<script>
  export default {
    emits: ['close']
  }
</script>
```

移除过滤器（filter）

> 过滤器虽然这看起来很方便，但它需要一个自定义语法，打破大括号内表达式是 “只是 JavaScript” 的假设，这不仅有学习成本，而且有实现成本！建议用方法调用或计算属性去替换过滤器。