一、单选题（共15道题，每题2分，共30分）

1、请选择如下代码中hello world文字的颜色。 c

<template>

<p class="outer"><span class="inner">hello world</span></p>

</template>

<style lang="scss" scoped>

.inner{ color: yellow; }

.outer .inner{ color: blue; }

.outer .inner{ color: green; }

p .inner{ color: red; }

</style>

A. yellow

B. blue

C. green

D. red

2、关于css的一些特性，选出表述错误的选项：a

A. 盒模型在页面所占的实际宽度是content+padding+border的宽度相加

B. 通过box-sizing可以改变盒模型，方便宽度的设置

C. flex布局中如果项目设置了flex：1，则再设置固定的width属性不会生效

D. 可以用overflow不为visible的方式形成BFC，解决margin穿透和重叠问题

3、下面哪种数组操作方法不会修改数组本身。 a

A. slice

B. splice

C. sort

D. unshift

4、如下代码中log的输出结果是： c

const obj1 = {a: 1, b: { c: 1, d: 1}};

const obj2 = {a: 2, b: { c: 2}}

const obj3 = Object.assign({}, obj1, obj2);

console.log('obj3: ', obj3);

obj1.b.c = 3;

console.log('obj3.b.c: ', obj3.b.c);

A. {a: 2, b: {c: 2}} ，3

B. {a: 2, b: {c: 2, d: 1}} ，3

C. {a: 2, b: {c: 2}} ，2

D. {a: 2, b: {c: 2, d: 1}} ，2

5、前后端联调过程中出现400状态码表明：b

A. 从客户端发送给服务器的请求被正常处理并返回

B. 请求参数错误

C. 请求不存在

D. 服务器遇到了意料不到的情况

6、关于异步编程方法，选出表述错误的选项：d

A. finally方法无论Promise链最终状态如何（fulfilled或rejected），都会执行

B. 使用Promise.all处理并发请求时，可以确保所有请求都完成后再进行下一步操作

C. async函数会隐式地返回一个Promise对象

D. 一个函数中的多个await表达式并行执行

7、关于本地存储，选出表述错误的选项：b

A. token用于验证用户登录状态，通常可存储在Cookie中

B. sessionStorage符合同源策略时，标签页间可共享

C. localStorage可以持久化存储，标签页间可共享

D. indexDB是浏览器内部的数据库，适合存储大量、复杂的数据

8、如下选出表述正确的选项：a

A. css3的transform变形与浏览器渲染主线程无关，这是其效率高的原因

B. 关于浏览器渲染，repaint重绘比reflow回流耗费性能

C. setTimeout方法能够精确控制函数在一定延迟后执行

D. requestAnimationFrame在页面被切换到后台时依然持续执行，保证动画连续性

9、基于事件循环原理，选择如下代码的执行结果：c

console.log(1);

async function async1() {

await async2();

console.log(2);

await async3();

console.log(3);

}

async function async2() {

console.log(4);

}

async function async3() {

console.log(5)

}

async1();

setTimeout(function() {

console.log(6)

}, 0);

new Promise(function(resolve, reject){

console.log(7);

resolve();

}).then(function(){

console.log(8);

});

console.log(9);

A. 1 4 2 5 3 9 7 8 6

B. 1 4 2 5 3 7 9 8 6

C. 1 4 7 9 2 5 8 3 6

D. 1 9 4 2 5 3 7 8 6

10、Vue中父子组件created和mounted生命周期调用顺序：d

A. 父created -> 父 mounted -> 子 created -> 子 mounted

B. 子 created -> 子 mounted -> 父created -> 父 mounted

C. 父 created -> 子 created -> 父 mounted -> 子 mounted

D. 父 created -> 子 created -> 子 mounted -> 父 mounted

11、Vue中如下哪种情况更适合使用watch而不是computed？b

A. 在模板中，如果需要进行较为复杂的逻辑判断或者计算

B. 当你需要根据某个数据的变化执行一系列非纯计算的操作，如API调用或延时操作

C. 当你需要缓存计算结果以提高性能

D. 当你需要根据一个或多个响应式依赖项计算一个衍生值

12、如下代码中当按钮被点击，increase方法执行的过程中，DOM被渲染了多少次，log打印出的内容是多少。 c

<template>

<button ref="counterRef" @click="increase">{{ count }}</button>

</template>

<script setup>

const counterRef = ref();

const count = ref(0);

function increase() {

for(let i=0; i<100; i++){

count.value++;

}

console.log(counterRef.value.innerHTML);

}

</script>

A. 100次， 0

B. 100次，100

C. 1次，0

D. 1次，100

13、关于Vue3中的ref与reactive，选出正确的表述：c

A. ref通常用于基础数据类型的响应式数据创建，因此其不能创建具备深层响应性的对象 

B. ref的使用比较麻烦，其在模板中使用时需要附加.value 

C. reactive有一定的局限性，它不能持有如 string、number 或 boolean 这样的原始类型

D. reactive对解构操作不友好，其解构出的数据都会丢失响应性连接

14、关于vue-router，选出错误的表述：b

A. 使用路由懒加载，当路由被访问的时候才动态加载对应组件，这样更加高效

B. $route可以通过push方法实现页面之间的跳转

C. 可以通过watch监听$route来响应路由的变化

D. 可以通过router的beforeEach路由守卫实现全局路由监听

15、关于TypeScript中的type和interface，选出错误的表述：d

A. type能够表示非对象类型，而interface只能表示对象类型（包括数组、函数等）

B. interface与type，都可以表示对象类型，两者往往可以换用

C. interface可以通过继承实现扩展，type通过交叉类型实现扩展

D. interface与type，都具备声明合并的能力，可以定义同名类型

二、简答题（共3道题，每题10分，共30分）

**1、Vue组件间通信方式有很多种，请例举父子间、祖孙间、兄弟组件间、任意组件间的通信方式，并简单说明如何使用。**

答: 父传子：

1. props：父组件通过props属性向子组件传递数据或者函数
2. v-model：是父子组件间实现双向数据绑定的语法糖，相当于传递:modelValue的props
3. $refs：父组件可以通过ref取到子组件的变量或方法，前提是子组件通过defineExpose将变量或方法抛出
4. 插槽：父组件可以通过插槽向子组件中传递dom内容

子传父：

1）props：子组件可以调用父组件传递的函数修改父组件中的数据

2）自定义事件：子组件通过emit触发事件，父组件监听这些事件来接收数据或执行操作

3）v-model：是父子组件间实现双向数据绑定的语法糖，vue3中使用defineModel修改父组件中绑定值，不需要触发update:modelValue事件来变更数据

4）$parent：子组件可以通过$parent取到父组件中变量或方法，前提是父组件通过defineExpose将变量或方法抛出，但是这种方式不推荐使用

5）作用域插槽：子组件可以通过作用域插槽，将子组件中的数据传递给父组件

祖传孙：

1）$attrs：$attrs包含了父组件传递给当前组件的非props属性的绑定，因此父组件作为中间组件可以通过v-bind=”$attrs”的方式将祖先组件中的props传递给孙子组件

此外父组件还可以通过v-on=”$listeners”使得孙子组件可以触发自定义事件到祖先组件

2）provide、inject：provide可以给任何组件后代提供数据或者方法，第一个参数是注入名，第二个参数是提供的值

inject用于注入上层组件提供的数据

兄弟间、任意组件间：

1. mitt：通过mitt创建EventBus事件总线对象，并将该对象绑定到Vue全局对象上，这样任何组件都能够访问到，其使用方法和组件间自定义事件相似
2. pinia：状态管理，简化了使用，不再有复杂的mutation和action分离，而是直接使用 actions 来改变状态

**2、Vue中进行逻辑复用的方式都有哪些，并对Vue2、Vue3两个版本中不同的处理方式进行优缺点比较。**

答:组件化：Vue中逻辑复用最基本的方式是封装方法函数和组件，其中组件可以将公共模板、样式、逻辑进行复用。这样大大的提高了开发效率，增强了代码的可维护性。

抽取： 当多个组件有一部分相同的处理逻辑时，我们可以将这部分进行抽取。在Vue2中使用mixins，在Vue3中使用组合式函数取而代之。

Vue2的mixins有如下的短板：

1. 不清晰的数据来源：当使用了多个 mixin 时，实例上的数据属性来自哪个 mixin 变得不清晰，这使追溯实现和理解组件行为变得困难。

而组合式函数中使用 ref + 解构模式，让属性的来源在消费组件时一目了然

1. 命名空间冲突：多个来自不同作者的 mixin 可能会注册相同的属性名，造成命名冲突。

若使用组合式函数，可以通过在解构变量时对变量进行重命名来避免相同的键名。

1. 隐式的跨 mixin 交流：多个 mixin 需要依赖共享的属性名来进行相互作用，这使得它们隐性地耦合在一起。

而一个组合式函数的返回值可以作为另一个组合式函数的参数被传入，像普通函数那样。

PS：如果是Vue2的老项目，可以通过使用@vue/composition-api插件使用组合式函数。

自定义指令：主要是为了重用涉及普通元素的底层 DOM 访问的逻辑。

插件：如果组件在多个项目中都使用，可以将其抽取为插件，插件能为Vue添加全局功能。

此外再说说模板复用的方法。有时候在一个文件中我们需要对部分模板内容进行复用，但是我们又不希望单独的抽取出组件，因为这样会丢失当前文件中的变量环境，还需要定义繁多的props和emits。这种情况可以尝试使用VueUse 里面的组件 [createReusableTemplate](https://vueuse.org/core/createreusabletemplate/#createreusabletemplate)实现。

**3、随着Vue升级，element组件库也升级到了element-plus，其相较于前一版本增加了虚拟化选择器和虚拟化表格组件，这些组件能够支持渲染千条、万条以上的数据，而不会造成页面卡顿。请简述这类虚拟化滚动组件的实现原理，或者说如果让你实现一个能够渲染万条以上的数据列表而不造成页面卡顿，你会如何实现。**

Element Plus 中新增的虚拟化选择器和虚拟化表格都是基于虚拟滚动（Virtual Scrolling）技术来优化大数据量下的性能表现，避免因渲染大量DOM节点导致的页面卡顿。其核心原理主要包括以下几个方面：

1）可视区域渲染

虚拟化组件仅渲染当前视口范围内可见的数据项，而非一次性渲染所有数据。这样，无论数据集有多大，实际渲染的DOM节点数量始终保持在一个较小的范围内，从而极大减少实际创建的DOM元素数量，显著减少浏览器的渲染负担。

2）DOM复用

列表中需要的DOM节点数量仅仅是能够覆盖视口范围内即可，还可以适当增加缓冲区域的条目数。当列表滚动时，始终是复用这些DOM节点，修改它们的位置或者内容进行复用。

3）计算逻辑

添加了一个滚动事件监听器，当用户滚动列表时，通过计算滚动位置、元素高度等信息，判断当前应该渲染哪一部分数据，进而更新列表项的位置和内容。

PS：如下是一个实现了简单虚拟滚动功能的列表代码，帮助大家理解上面的三点原理：

// 假设我们有一个很长的列表

const list = Array.from({ length: 10000 }, (\_, i) => `Item ${i}`);

// 滚动容器

const container = document.querySelector('.list-box');

// 可视区域高度

const visibleHeight = container.clientHeight;

// 列表项的高度

const itemHeight = 50;

// 可视区域渲染条数

const visibleCount = Math.ceil(visibleHeight / itemHeight) + 2;

// 创建一个内容容器

const content = document.createElement('div');

content.style.position = 'relative';

content.style.height = `${list.length \* itemHeight}px`;

container.appendChild(content);

// 创建可视区域内的复用DOM列表项

const items = Array.from({ length: visibleCount }, () => {

const item = document.createElement('div');

item.style.position = 'absolute';

item.style.height = `${itemHeight}px`;

content.appendChild(item);

return item;

});

// 更新可视区域内的列表项

function updateItems() {

const start = Math.floor(container.scrollTop / itemHeight);

for (let i = 0; i < visibleCount; i++) {

const item = items[i];

const index = start + i;

item.style.transform = `translateY(${index \* itemHeight}px)`

item.textContent = list[index];

}

}

// 添加滚动事件监听器

container.addEventListener('scroll', updateItems);

// 初始化可视区域内的列表项

updateItems();

三、编程题（共3道题，1、2题每题10分，第3题20分，共40分）

**1、在处理业务的时候，我们经常需要对数组排序和获取极值。不使用第三方工具库，实现下面的功能要求。**

**1）请对如下数组中元素从小到大排序。**

**const arr = [10, 5, 8, 3, 6, 2, 4, 7, 9, 1]**

**1）**

arr.sort((a, b) => a - b);

2）

方法一：

arr.sort((a, b) => b.value - a.value);

const maxItem = arr[0];

方法二：

const maxItem = arr.reduce((a, b) => {

return a.value > b.value?a:b;

});

2）请根据value属性获取如下对象数组中的最大值{ name: 'd', value: 10 }。

const arr = [

{ name: 'a', value: 5 },

{ name: 'b', value: 2 },

{ name: 'c', value: 8 },

{ name: 'd', value: 10 },

{ name: 'e', value: 4 },

{ name: 'f', value: 3 }

];

**2、请封装一个函数方法，对后端接口返回的以KB为单位的数值进行单位转换，格式化为MB, GB, TB, PB单位的数值，如：10000000 转换为 ‘9.54GB’。**

function convertUnit(sizeKb, fixed = 2){

const units = ['KB', 'MB', 'GB', 'TB', 'PB'];

let index = 0;

while(sizeKb > 1024 && index < units.length - 1){

sizeKb /= 1024;

index++;

}

return sizeKb.toFixed(fixed) + units[index];

}

3、在开发一个内容展示应用时，需要频繁展示不同样式和内容的卡片列表，如项目摘要、产品展示等。为了提高开发效率和维护性，决定创建一个可配置的卡片列表组件。

需求说明：

1）请设计并实现一个Vue卡片列表组件——CardList.vue，该组件需求如下：

1. 动态内容与样式：组件能够根据传入的数据和配置参数动态渲染一系列卡片，每张卡片展示的内容和样式可通过插槽自定义。
2. 基础配置：至少支持以下基础配置项：

* cardsData：一个对象数组，每个对象代表一张卡片的内容数据。
* listLayout：定义列表的布局方式，如'list'或'grid'。list为平铺的列表，每个卡片占一行，grid为网格布局。

1. 响应式设计：列表具备基本的响应式，能够在不同屏幕尺寸下自动调整卡片布局。
2. 筛选：列表顶部添加筛选输入框，能够随着输入内容触发列表查询，并且需要进行输入防抖处理，防抖方法自行实现。
3. 不要求列表的动态加载和分页功能
4. 只需写出关键样式代码
5. 优先使用Vue3组合式Api语法

2）写出使用上面卡片列表组件的使用示例App.vue。

卡片列表数据：

[

{ title: 'Card 1', summary: 'Summary of card 1' },

{ title: 'Card 2', summary: 'Summary of card 2' },

{ title: 'Card 3', summary: 'Summary of card 3' },

{ title: 'Card 4', summary: 'Summary of card 4' },

{ title: 'Card 5', summary: 'Summary of card 5' },

{ title: 'Card 6', summary: 'Summary of card 6' },

{ title: 'Card 7', summary: 'Summary of card 7' }

]

PS：代码中上述数据用const cards = […];代替即可。

1）CardList.vue：

<template>

<div class="card-filter">

<el-input v-model="filter" placeholder="请输入搜索关键字" />

</div>

<div class="card-list" :class="{ grid: listLayout == 'grid' }">

<el-card v-for="(item, index) in cards" :key="'card-' + index">

<slot :data="item"></slot>

</el-card>

</div>

</template>

<script setup>

const props = defineProps({

cardsData: {

type: Array,

default: () => ([])

},

listLayout: {

type: String,

default: "list",

validator: function (value) {

return ["list", "grid"].includes(value);

},

},

});

const cards = ref(props.cardsData);

function debounce(fn, delay) {

let timer;

return function(...args) {

clearTimeout(timer);

timer = setTimeout(function() {

fn.apply(this, args);

}, delay);

}

}

function searchHandler(val) {

cards.value = props.cardsData.filter((item) => {

return item.title.includes(val);

});

}

const searchDebounce = debounce(searchHandler, 300);

const filter = ref("");

watch(filter, (val) => {

searchDebounce(val);

})

</script>

<style lang="scss" scoped>

.card-list {

&.grid {

display: grid;

grid-template-columns: repeat(auto-fit, 300px);

grid-column-gap: 20px;

}

}

</style>

2）App.vue

<template>

<card-list

:cards-data="cards"

list-layout="grid"

#default="{ data }"

>

<div class="card">

<h2>{{ data.title }}</h2>

<p>{{ data.summary }}</p>

</div>

</card-list>

</template>

<script setup>

import CardList from "./components/CardList.vue";

const cards = [

{ id: "card-1", title: "Card 1", summary: "Summary of card 1" },

{ id: "card-2", title: "Card 2", summary: "Summary of card 2" },

{ id: "card-3", title: "Card 3", summary: "Summary of card 3" },

{ id: "card-4", title: "Card 4", summary: "Summary of card 4" },

{ id: "card-5", title: "Card 5", summary: "Summary of card 5" },

{ id: "card-6", title: "Card 6", summary: "Summary of card 6" },

{ id: "card-7", title: "Card 7", summary: "Summary of card 7" },

];

</script>