最全 Vue 知识点（基础到进阶覆盖vue3）

你知道Vue3.x响应式数据原理吗？

- **Vue3.x改用Proxy替代Object.defineProperty。**
- 因为Proxy可以直接监听对象和数组的变化，并且有多达13种拦截方法。并且作为新标准将受到浏览器厂商重点持续的性能优化。
- Proxy只会代理对象的第一层，Vue3是怎样处理这个问题的呢？
    - 判断当前Reflect.get的返回值是否为Object，如果是则再通过reactive方法做代理， 这样就实现了深度观测。
    - 监测数组的时候可能触发多次get/set，那么如何防止触发多次呢？我们可以判断key是否为当前被代理对象target自身属性，也可以判断旧值与新值是否相等，只有满足以上两个条件之一时，才有可能执行trigger。

Proxy 与 Object.defineProperty 优劣对比

- Proxy 的优势如下:
    - Proxy 可以直接监听对象而非属性；
- Proxy 可以直接监听数组的变化；
    - Proxy 有多达 13 种拦截方法,不限于 apply、ownKeys、deleteProperty、has 等等是 Object.defineProperty 不具备的；
    - Proxy 返回的是一个新对象,我们可以只操作新的对象达到目的,而 Object.defineProperty 只能遍历对象属性直接修改；
    - Proxy 作为新标准将受到浏览器厂商重点持续的性能优化，也就是传说中的新标准的性能红利；
- Object.defineProperty 的优势如下:
    - 兼容性好，支持 IE9，而 Proxy 的存在浏览器兼容性问题,而且无法用 polyfill 磨平，因此 Vue 的作者才声明需要等到下个大版本(3.0)才能用 Proxy 重写。

什么情况下使用 Vuex？

- 如果应用够简单，最好不要使用 Vuex，一个简单的 store 模式即可
- 需要构建一个中大型单页应用时，使用Vuex能更好地在组件外部管理状态

Vuex和单纯的全局对象有什么区别？

- Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
- 不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

为什么 Vuex 的 mutation 中不能做异步操作？

- Vuex中所有的状态更新的唯一途径都是mutation，异步操作通过 Action 来提交 mutation实现，这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

- 每个mutation执行完成后都会对应到一个新的状态变更，这样devtools就可以打个快照存下来，然后就可以实现 time-travel 了。如果mutation支持异步操作，就没有办法知道状态是何时更新的，无法很好的进行状态的追踪，给调试带来困难。

新增：vuex的action有返回值吗？返回的是什么？

- store.dispatch 可以处理被触发的 action 的处理函数返回的 Promise，并且 store.dispatch 仍旧返回 Promise
- Action 通常是异步的，要知道 action 什么时候结束或者组合多个 action以处理更加复杂的异步流程，可以通过定义action时返回一个promise对象，就可以在派发action的时候就可以通过处理返回的 Promise处理异步流程

> 一个 store.dispatch 在不同模块中可以触发多个 action 函数。在这种情况下，只有当所有触发函数完成后，返回的 Promise 才会执行。

新增：为什么不直接分发mutation,而要通过分发action之后提交 mutation变更状态

- mutation 必须同步执行，我们可以在 action 内部执行异步操作
- 可以进行一系列的异步操作，并且通过提交 mutation 来记录 action 产生的副作用（即状态变更）

常规篇

computed 和 watch 的区别和运用的场景？

- computed：是计算属性，依赖其它属性值，并且 computed 的值有**缓存**，只有它**依赖的属性值**发生改变，下一次获取 computed 的值时才会重新计算 computed 的值；
- watch：没有缓存性，更多的是「**观察**」的作用，类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作；当我们需要深度监听对象中的属性时，可以打开deep：true选项，这样便会对对象中的每一项进行监听
- **运用场景**：
  - 当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；
  - 当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用watch选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

Vue2.x组件通信有哪些方式？

- 父子组件通信
  - 事件机制(**父->子props,子->父 `$on、$emit`)
  - 获取父子组件实例 `$parent、$children`
  - Ref 获取实例的方式调用组件的属性或者方法
  - Provide、inject (不推荐使用，组件库时很常用)
- 兄弟组件通信

  > Vue.prototype.`$bus` = new Vue

  - **Vuex**
  - **eventBus** 这种方法通过**一个空的 Vue实例作为中央事件总线（事件中心）**，用它来触发事件和监听事件，从而实现任何组件间的通信，包括父子、隔代、兄弟组件
- 跨级组件通信

- Vuex
- `$attrs、$listeners`
- Provide、inject

为什么 v-for 和 v-if 不建议用在一起

- 当 v-for 和 v-if 处于同一个节点时，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。如果要遍历的数组很大，而真正要展示的数据很少时，这将造成很大的性能浪费
- 这种场景建议使用 computed，先对数据进行过滤

组件中的data为什么是一个函数？

- 一个组件被复用多次的话，也就会创建多个实例。本质上，这些实例用的都是同一个构造函数。
- 如果data是对象的话，对象属于引用类型，会影响到所有的实例。所以为了保证组件不同的实例之间data不冲突，data必须是一个函数。

子组件为什么不可以修改父组件传递的Prop？/怎么理解vue的单向数据流？

- Vue提倡单向数据流,即父级props的更新会流向子组件,但是反过来则不行。
- 这是为了防止意外的改变父组件状态，使得应用的数据流变得难以理解。
- 如果破坏了单向数据流，当应用复杂时，debug 的成本会非常高。

v-model是如何实现双向绑定的？

- v-model是用来在表单控件或者组件上创建双向绑定的
- 他的本质是v-bind和v-on的语法糖
- 在一个组件上使用v-model，默认会为组件绑定名为value的prop和名为input的事件

nextTick的实现原理是什么？

- 在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后立即使用 nextTick 来获取更新后的 DOM。
- nextTick主要使用了**宏任务**和**微任务**。
- 根据执行环境分别尝试采用Promise、MutationObserver、setImmediate，如果以上都不行则采用setTimeout定义了一个异步方法，多次调用nextTick会将方法存入队列中，通过这个异步方法清空当前队列。

Vue不能检测数组的哪些变动？Vue 怎么用 `vm.$set()` 解决对象新增属性不能响应的问题？

- Vue 不能检测以下数组的变动：
  - 第一类问题

    `// 法一：Vue.set Vue.set(vm.items, indexOfItem, newValue) // 法二：Array.prototype.splice vm.items.splice(indexOfItem, 1, newValue) 复制代码`
  - 第二类问题，可使用 splice：

    `vm.items.splice(newLength) 复制代码`
  - 当你利用索引直接设置一个数组项时，例如：vm.items[indexOfItem] = newValue
  - 当你修改数组的长度时，例如：vm.items.length = newLength
  - 解决办法：
- vm.`$set` 的实现原理是：

    - 如果目标是数组，直接使用数组的 splice 方法触发相应式；
    - 如果目标是对象，会先判读属性是否存在、对象是否是响应式，最终如果要对属性进行响应式处理，则是通过调用 defineReactive 方法进行响应式处理（ defineReactive 方法就是 Vue 在初始化对象时，给对象属性采用 Object.defineProperty 动态添加 getter 和 setter 的功能所调用的方法）

Vue事件绑定原理是什么？

- 原生事件绑定是通过addEventListener绑定给真实元素的，组件事件绑定是通过Vue自定义的`$on`实现的。

说一下虚拟Dom以及key属性的作用

- 由于在浏览器中操作DOM是很昂贵的。频繁的操作DOM，会产生一定的性能问题。这就是虚拟Dom的产生原因。
- Virtual DOM本质就是用一个**原生的JS对象去描述一个DOM节点。**是对真实DOM的一层抽象。(也就是源码中的VNode类，它定义在src/core/vdom/vnode.js中。)
- 虚拟 DOM 的实现原理主要包括以下 3 部分：
    - 用 JavaScript 对象模拟真实 DOM 树，对真实 DOM 进行抽象；
    - diff 算法 — 比较两棵虚拟 DOM 树的差异；
    - pach 算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM 树。
- key 是为 Vue 中 vnode 的唯一标记，通过这个 key，我们的 diff 操作可以更准确、更快速
    - 更准确：因为带 key 就不是就地复用了，在 sameNode 函数a.key === b.key对比中可以避免就地复用的情况。所以会更加准确。
    - 更快速：利用 key 的唯一性生成 map 对象来获取对应节点，比遍历方式更快

为什么不建议用index作为key?

- 不建议 用index 作为 key，和没写基本上没区别，因为不管你数组的顺序怎么颠倒，index 都是 0, 1, 2 这样排列，导致 Vue 会复用错误的旧子节点，做很多额外的工作

生命周期篇

说一下你对Vue的生命周期的理解

- 简单回答
    - beforeCreate、created、beforeMount、mounted、beforeUpdate、updated、beforeDestroy、destroyed。
    - keep-alive 有自己独立的钩子函数 activated 和 deactivated。
- 复杂回答

| 生命周期

发生了什么
beforeCreate
created
beforeMount
mounted
beforeUpdate
updated
beforeDestroy
destroyed
activited keep-alive 专属
deactivated keep-alive 专属

Vue中组件生命周期调用顺序是什么样的？

- 组件的调用顺序都是先父后子,渲染完成的顺序是先子后父。
- 组件的销毁操作是先父后子，销毁完成的顺序是先子后父。

在什么阶段才能访问操作DOM？

在钩子函数 mounted 被调用前，Vue 已经将编译好的模板挂载到页面上，所以在 mounted 中可以访问操作 DOM。

你的接口请求一般放在哪个生命周期中？

- 可以在钩子函数 created、beforeMount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的数据进行赋值。
- 但是推荐在 created 钩子函数中调用异步请求，因为在 created 钩子函数中调用异步请求有以下优点：
    - 能更快获取到服务端数据，减少页面loading 时间；
    - ssr不支持 beforeMount 、mounted 钩子函数，所以放在 created 中有助于一致性；

路由篇

vue路由hash模式和history模式实现原理分别是什么，他们的区别是什么？

- hash 模式：
    - #后面 hash 值的变化，不会导致浏览器向服务器发出请求，浏览器不发出请求，就不会刷新页面
    - 通过监听 **hashchange** 事件可以知道 hash 发生了哪些变化，然后根据 hash 变化来实现更新页面部分内容的操作。
- history 模式：
    - history 模式的实现，主要是 HTML5 标准发布的两个 API，**pushState** 和**replaceState**，这两个 API 可以在改变 url，但是不会发送请求。这样就可以监听 url 变化来实现更新页面部分内容的操作

- 区别
    - url 展示上，hash 模式有“#”，history 模式没有
    - 刷新页面时，hash 模式可以正常加载到 hash 值对应的页面，而 history 没有处理的话，会返回 404，一般需要后端将所有页面都配置重定向到首页路由
    - 兼容性，hash 可以支持低版本浏览器和 IE。

路由懒加载是什么意思？如何实现路由懒加载？

- 路由懒加载的含义：把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件
- 实现：结合 Vue 的异步组件和 Webpack 的代码分割功能

```
1. 可以将异步组件定义为返回一个 Promise 的工厂函数 (该函数返回的 Promise 应该 resolve 组件本身)
```

`const Foo = () => Promise.resolve({ /* 组件定义对象 */ }) 复制代码`

```
2. 在 Webpack 2 中，我们可以使用动态 import语法来定义代码分块点 (split point)
```

`import( ./Foo.vue ) // 返回 Promise 复制代码`

- 结合这两者，这就是如何定义一个能够被 Webpack 自动代码分割的异步组件

`const Foo = () => import( ./Foo.vue ) const router = new VueRouter({ routes: [ { path: /foo , component: Foo } ]}) 复制代码`

- 使用命名 chunk，和webpack中的魔法注释就可以把某个路由下的所有组件都打包在同个异步块 (chunk) 中

```
chunkconst Foo = () => import(/* webpackChunkName: "group-foo" */ ./Foo.vue ) 复制代码
```

Vue-router 导航守卫有哪些

- 全局前置/钩子：beforeEach、beforeResolve、afterEach
- 路由独享的守卫：beforeEnter
- 组件内的守卫：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

在 Vue 实例中编写生命周期 hook 或其他 option/properties 时，为什么不使用箭头函数？

- 箭头函数自已没有定义 this 上下文中。
- 当你在 Vue 程序中使用箭头函数 (=>) 时，this 关键字病不会绑定到 Vue 实例，因此会引发错误。所以强烈建议改用标准函数声明。

说说你对keep-alive组件的了解

- keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染 ，其有以下特性：
    - 一般结合路由和动态组件一起使用，用于缓存组件；
    - 提供 include 和 exclude 属性，两者都支持字符串或正则表达式， include 表示只有名称匹配的组件会被缓存，exclude 表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include 高；
    - 对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。

说说你对SSR的了解

- SSR也就是服务端渲染，也就是将Vue在客户端把标签渲染成HTML的工作放在服务端完成，然后再把html直接返回给客户端
- SSR的优势
    - 更好的SEO
    - 首屏加载速度更快
- SSR的缺点
    - 开发条件会受到限制，服务器端渲染只支持beforeCreate和created两个钩子
    - 当我们需要一些外部扩展库时需要特殊处理，服务端渲染应用程序也需要处于Node.js的运行环境
    - 更多的服务端负载

你都做过哪些Vue的性能优化？

- 编码阶段
    - 尽量减少data中的数据，data中的数据都会增加getter和setter，会收集对应的watcher
    - v-if和v-for不能连用
    - 如果需要使用v-for给每项元素绑定事件时使用事件代理
    - SPA 页面采用keep-alive缓存组件
    - 在更多的情况下，使用v-if替代v-show
    - key保证唯一
    - 使用路由懒加载、异步组件
    - 防抖、节流
    - 第三方模块按需导入
    - 长列表滚动到可视区域动态加载
    - 图片懒加载
- SEO优化
    - 预渲染
    - 服务端渲染SSR
- 打包优化
    - 压缩代码
    - Tree Shaking/Scope Hoisting
    - 使用cdn加载第三方模块
    - 多线程打包happypack
    - splitChunks抽离公共文件
    - sourceMap优化
- 用户体验
    - 骨架屏
    - PWA
    - 还可以使用缓存(客户端缓存、服务端缓存)优化、服务端开启gzip压缩等。

vue2.x中如何监测数组变化？

- 使用了函数劫持的方式，重写了数组的方法，Vue将data中的数组进行了原型链重写，指向了自己定义的数组原型方法，当调用数组api时，可以通知依赖更新。
- 如果数组中包含着引用类型，会对数组中的引用类型再次递归遍历进行监控。这样就实现了监测数组变化。

对于即将到来的 vue3.0 特性你有什么了解的吗？

- 监测机制的改变
    - 3.0 将带来基于代理 Proxy的 observer 实现，提供全语言覆盖的反应性跟踪。
    - 消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的很多限制：
- 只能监测属性，不能监测对象
    - 检测属性的添加和删除；
    - 检测数组索引和长度的变更；
    - 支持 Map、Set、WeakMap 和 WeakSet。
- 模板
    - 模板方面没有大的变更，只改了作用域插槽，2.x 的机制导致作用域插槽变了，父组件会重新渲染，而 3.0 把作用域插槽改成了函数的方式，这样只会影响子组件的重新渲染，提升了渲染的性能。
    - 同时，对于 render 函数的方面，vue3.0 也会进行一系列更改来方便习惯直接使用 api 来生成 vdom 。
- 对象式的组件声明方式
    - vue2.x 中的组件是通过声明的方式传入一系列 option，和 TypeScript 的结合需要通过一些装饰器的方式来做，虽然能实现功能，但是比较麻烦。
    - 3.0 修改了组件的声明方式，改成了类式的写法，这样使得和 TypeScript 的结合变得很容易
- 其它方面的更改
    - 支持自定义渲染器，从而使得 weex 可以通过自定义渲染器的方式来扩展，而不是直接 fork 源码来改的方式。
    - 支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组建内容）组件，针对一些特殊的场景做了处理。
    - 基于 tree shaking 优化，提供了更多的内置功能