Vue2相关面试题

一、vue2和react的区别

相同点：

1. 维护核心库，其他功能如路由和全局状态管理交给生态来做；

2. 都有自己的构建工具

3. 都使用了虚拟DOM来提高重绘性能；

4. 都有props的概念，允许组件间的数据传递；

5. 实现组件化应用，提高复用性

不同点：

1. 数据流：Vue默认支持数据双向绑定，而React一直提倡单向数据流

2. vue默认对虚拟DOM具有优化会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树；React每当应用的状态被改变时，全部子组件都会重新渲染需要手动通过PureComponent/shouldComponentUpdate这个生命周期方法来进行控制

3. vue使用类似常规的HTML模版，react推荐你所有的模板使用JSX书写。

1. Vue2通过Object.defineProperty监听数据变化,vue3通过proxy，具备比较好的性能React监听数据是通过比较引用的方式，需要手动优化

5. react使用hock拓展，Vue2使用mixins，vue3使用hooks

二、vue2和vue3区别

答：生命周期钩子不同、数据双向绑定原理不同、定义变量和方法的方式不同、指令和插槽的使用不同(v-if和v-for不在相互冲突，移除filter、使用v-slot)、vue3支持多个根节点

1. Vue优点 - 轻量级框架大小只有几十kb、易于理解和学习、上手成本低、运行速度更快
2. 什么是MVVM？

答：Model数据模型，View视图层，ViewModel视图数据层，MVVM模式不需要用户手动渲染DOM元素，而是直接将数据绑定到viewModel上面，会自动渲染数据到页面当中，试图变化会通知viewModel层更新数据，viewModel就是我们MVVM模式当中的桥梁，向下负责转换 Model 中的数据对象，向上为View做服务，解耦了 View 层和 Model 层

五、MVVM的优点：降低代码耦合，提⾼视图或者逻辑的重⽤性、利⽤双向绑定数据更新后视图⾃动更新

六、MVVM缺点：调试Bug需要依赖一些工具，ViewModel的构建和维护的成本都会⽐较⾼会影响性能

七：库：将代码集合成一个产品，我们调用库中的方法来实现自己的功能

八、框架是为了解决一类问题而开发的产品，框架是我们在指定的位置编写好代码，框架帮我们调用

九、Vue 单页应用与多页应用的区别

1. SPA单页面应用指只有一个主页面的应用，一开始只需要加载一次js、css等相关资源。所有内容都包含在主页面对每一个功能模块组件化。单页应用跳转切换相关组件刷新局部资源。

2. MPA多页面应用指有多个独立页面的应用，每个页面必须重复加载js、css等相关资源。多页应用跳转，需要整页资源刷新。

十、请说一下vue响应式数据的理解？（Vue的响应式原理）

1. 通过Object.defineProperty/proxy对data的getter/setter方法进行拦截，利用发布订阅者模式，在getter方法中进行订阅，在setter方法中通知，让所有订阅者完成响应。data的每一个属性都新建一个订阅中心作为发布者，对于watcher、computed、template/render作为订阅者，watcher会直接订阅观察监听的属性，对于computed和template/render，如果内部执行获取了data的某一个属性值，就会执行这个属性的getter方法，然后自动完成对该属性的订阅，当属性被修改后，就会执行这个属性的setter方法，从而完成这个属性的发布通知，通知所有的订阅者进行更新。

十一、Vue 是如何实现数据双向绑定的？

数据双向绑定是数据变化更新视图视图变化更新数据，View变化更新Data可以通过事件监听的方式来实现，主要通过以下 4 个步骤来实现数据双向绑定的：

1. 实现一个监听器 Observer：对数据对象进行遍历，包括子属性对象的属性，利用 Object.defineProperty() 对属性都加上 setter 和 getter。这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化。

1. 2. compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图。

3. Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是: ①在自身实例化时往属性订阅器(dep)里面添加自己 ②自身必须有一个update()方法 ③待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调。

4. 订阅器 Dep采用发布-订阅设计模式，用来收集订阅者 Watcher，对监听器 Observer 和 订阅者 Watcher 进行统一管理。

MVVM作为数据绑定的入口整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

十二、vue中是如何检测数组变化？

vue2当中通过函数劫持的方式重写了数组的方法，利用原型链的特性，每次调用这些方法的时候实际上走的是我们自己重写的方法，vue3当中改用proxy，可以直接监听数组的变化

十三、使用 Object.defineProperty来进行数据劫持有什么缺点？

1. 对象默认只监控自带的属性，新增的属性响应式不生效 (层级过深，性能差)

2. 数组通过索引进行修改 或者 修改数组的长度，响应式不生效

为了解决以上问题，vm.$set -> 修改数组内部使用的是splice方法 和vm.$delete

十四、Proxy与Object.defineProperty优劣对比

1. Proxy 可以直接监听对象而非属性，可以直接监听数组的变化

2. Proxy 有多达13种拦截方法，不限于 apply、ownKeys、deleteProperty

3. Proxy 返回的是一个新对象我们可以只操作新的对象而Object.defineProperty只能遍历对象属性修改

4. Proxy 如果对象内部要全部递归代理，则Proxy可以只在调用时递归，而Object.defineProperty需要在一开始就全部递归，Proxy性能优于Object.defineProperty

5. Object.defineProperty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

十五、怎么理解vue的单向数据流？

父子组件之间的通信是单向的，是为了防止子组件意外的改变父组件的状态使得数据流难以理解，每次父级组件发生更新时，子组件中所有的props都会更新，子组件通过$emit派发一个自定义事件，父组件接收到后，由父组件修改数据

十六、vue数据的丢失 - 初始化时没有定义数据，之后更新的数据是无法触发页面渲染更新的

十七、vue 中给data中的对象属性添加一个新的属性时会发生什么？如何解决？

不会发生试图更新；需要使用$set方法相当于手动的去把新增加的对象属性处理成一个响应式的属性，此时视图也会跟着改变了。

十八、data中某一个属性的值发生改变后，视图会立即同步执行重新渲染吗？

不会立即同步执行重新渲染，Vue实现响应式并不是数据发生变化之后DOM立即变化，而是按一定的策略进行DOM的更新。在更新DOM时是异步的，监测到数据变化Vue 将开启一个队列进行缓冲，在同一事件循环中发生的所有数据变更。如果同一个watcher被多次触发，只会入到队列中一次，因为在缓冲时会去除重复数据，避免不必要的计算和DOM操作，在下一个的事件循环中会刷新队列并执行实际的工作。

十九、在data上面定义了一个$name属性，可以通过Vue.$name拿到吗？

不可以，因为vue内部将所有$开头的属性都默认为是vue本身内置的属性

二十、直接给一个数组项赋值，Vue 能检测到变化吗？

不能，数组考虑性能原因没有用defineProperty对数组的每一项进行拦截，而是选择重写数组方法，在Vue中修改数组的索引和长度是无法监控到的。需要通过上面的方法修改数组才会触发数组对应的watcher进行更新。数组中如果是对象数据类型也会进行递归劫持。可以通过Vue.$set来进行处理, 其核心内部用的是splice方法，针对修改数组的长度使用\*\*vm.items.splice(newLength)\*\*解决

二十一、 Vue中封装的数组方法有哪些，其如何实现页面更新

push,shift,pop,splice,unshift,sort,reverse；首先获取到这个数组的\_\_ob\_\_即Observer对象，如果有新的值就调用observeArray继续对新的值观察变化（也就是通过target.\_\_proto\_\_ == arrayMethods来改变了数组实例的原型）然后手动调用notify通知渲染watcher执行update

二十二、Vue怎么用vm.$set解决数组对象新增属性不能响应 - 通过调用重写的数组splice方法实现

二十三、给响应式数据添加标识为什么不可以直接使用value.\_\_ob\_\_ = this

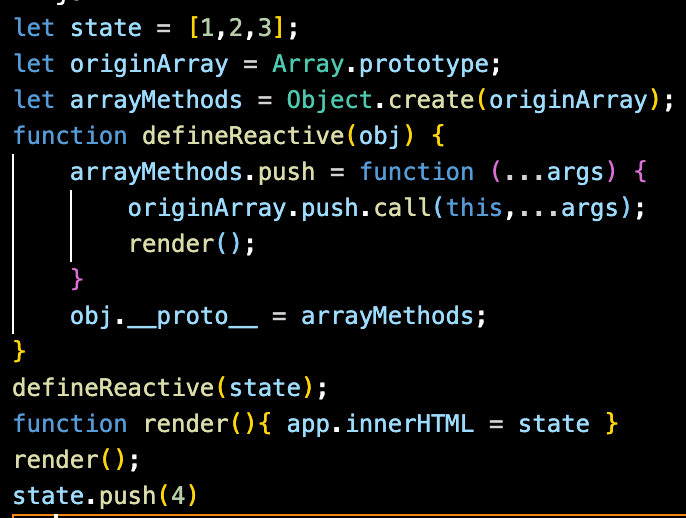
因为value是一个被观测得响应式的值如果这样赋值就会触发Object.defineProperty方法set方法，此时无限递归会陷入死循环，通过Object.defineProperty方法添加\_\_ob\_\_属性这个属性是不能被枚举不能被循环出来的，所以不会有影响

二十四、vue的构造函数为什么没有使用class语法糖

vue构造函数的方法很多，如果使用class进行维护会篇幅很大，对于重写方法的行为也不是很友好，所以还是采用了构造函数的方式，在其原型上增加方法进行拓展

二十五、定义响应式数据的时候在defineReactive的set方法内部，不直接value[key]赋值？会陷入死循环

二十六、手动实现一个proxy的响应式 二十七、Vue是如何重写数组方法的？

二十八、vue为什么采用异步渲染？

vue的数据更新是采用组件级别的更新策略，如果同一个组件内部的数据多个变化不采用异步更新的策略，每次更新数据视图都会跟着一起更新会比较消耗性能，vue会将同一个组件的watcher的更新过滤在一起在合适的时机更新，主要操作是：dep.notify通知watcher进行数据更新，然后依次调用watcher的update方法，通过queueWatcher方法将多个属性依赖的相同的watcher整合成一个（每一个watcher都有一个id相同id的整合在一起主要是预防多次更新）最后通过nextTick方法异步清空watcher队列，vue会在数据更新了之后，再去异步更新视图，提高了性能

二十九、Vue中模板编译原理？

模板编译：vue中的模板template无法被浏览器解析并渲染，因为不是正确的HTML语法，所有需要将template转化成一个JavaScript函数，这样浏览器就可以执行这一个函数并渲染出对应的HTML元素，就可以让视图跑起来了。模板编译分为解析parse、优化optimize、生成generate，最终生成可执行函数render。

1. parse阶段：使用大量正则表达式对template字符串进行解析将标签指令属性等转化为抽象语法树AST。
2. optimize阶段：遍历AST找到其中的一些静态节点并进行标记方便在页面重渲染的时候进行diff比较时，直接跳过这一些静态节点，优化runtime的性能。3. generate将最终AST转化为render函数字符串。

三十、template预编译是什么？

模板编译只会在组件实例化的时候编译一次生成渲染函数之后在也不会进行编译。因此，编译对组件的 runtime 是一种性能损耗。而模板编译的目的仅仅是将template转化为render function，这个过程，正好可以在项目构建的过程中完成，这样可以让实际组件在 runtime 时直接跳过模板渲染，进而提升性能，这个在项目构建的编译template的过程，就是预编译。

三十一、vue 是怎么解析template的? template会变成什么?

源码会执行 compileToFunctions 将template转化为render函数，调用parse方法将template转化为ast对静态节点做优化，generate将ast抽象语法树编译成 render字符串并将静态部分放到 staticRenderFns 中，最后通过 new Function(``render``) 生成render函数

三十二、模板引擎的实现原理 - 就是\*\*new Function + with\*\*来进行实现的

三十三、computed 什么时候初始化

vue创建实例的时候会初始化各种选项其中initState方法中会初始化`props、methods、data、computed、以及watch`，其中initComputed就是初始化computed

三十四、computed 的缓存是怎么做的？

computed本身就是一个计算属性类型的watcher，缓存是通过控制watcher实例对象的dirty属性做到的，computed将lazy赋值dirty给一个初始值，让watcher控制缓存的任务开始，如果 computed依赖的数据变化，dirty=true，调用evalute重新计算然后更新缓存值watcher.value；computed的A依赖B，就会将B收集到A的watcher种，B变了就会通知A更新，就会判断dirty = true重新计算否则走缓存

三十五、computed是怎么可以直接使用实例访问到的

在defineComputed函数当中通过`Object.defineProperty(target, key, sharedPropertyDefinition)`将computed的所有属性都定义在了vm上面，所以可以直接使用实例访问到的

三十六、computed是怎样进行计算的

computed在初始化watcher实例时，并没有计算值，`watcher.value = undefined`，只有调用了`computed`的属性时，控制`watcher`实例对象的`dirty`设置为true才会发生计算。在计算的过程中，computed-watcher.evaluted 被调用，进而 computed-watcher.get 被调用，Dep.target 被设置为 computed-watcher，旧值 页面 watcher 被缓存起来。computed 计算会读取 data，此时 data 就收集到 computed-watcher，同时 computed-watcher 也会保存到 data 的依赖收集器 dep。computed 计算完毕，释放Dep.target，并且Dep.target 恢复上一个watcher（页面watcher），返回value的更新值

## 5. computed 的 月老身份的来源

1. 页面更新，读取 computed 的时候，Dep.target 会设置为 页面 watcher。

2. computed被读取，createComputedGetter包装的函数触发，第一次会进行计算

3. computed-watcher.evaluted 被调用，进而computed-watcher.get被调用，Dep.target被设置为computed-watcher，旧值页面 watcher 被缓存起来。computed 计算会读取 data，此时 data 就收集到 computed-watcher，同时 computed-watcher 也会保存到 data 的依赖收集器 dep（用于下一步）。computed 计算完毕，释放Dep.target，并且Dep.target 恢复上一个watcher（页面watcher）

4. 手动 watcher.depend， 让 data 再收集一次 Dep.target，于是 data 又收集到 恢复了的页面watcher，从而使得页面A与数据C关联上

## 5. computed与watch的区别

\*\*区别\*\*

+ 计算属性computed和监听器watcher都可以观察属性的变化从而做出响应

+ computed本质上就是一个watcher，与watch最大的区别就在于computed更多的是作为缓存功能的观察者，它可以将一个或者多个data的属性进行复杂的计算生成一个新的值，提供给渲染函数使用，当依赖发生变化时，computed不会立即重新计算生成新的值，而是先标记为脏数据，当下次computed被获取的时候，才会重新计算并返回；而监听器watch并不具备缓存性，监听器watch提供一个监听函数，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

\*\*computed与watch的使用场景\*\*

+ 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。

## 6. 在使用计算属性时，函数名和 data 数据源中的数据可以同名吗？

不可以重名，不仅仅是计算属性和 data，其他的如 props，methods 都不可以重名，因为 Vue 会把这些属性挂载在组件实例上，直接使用 this 就可以访问，如果重名就会导致冲突。在组件初始化的时候，会去查你定义的 `computed` 的 key 值在 `data `和 `props` 中有没有存在，如果有冲突的话就会报错了

## 7. watch的原理

watch用于监控用户的data变化，数据变化后会触发对应的watch的回调方法,vue当中的watch属性本质上其实调用的是vue暴露在实例上面的`$watch`方法，`$watch`内部就是创建一个`watcher`的过程，如果用户传进来的还有一些特殊的操作做一些特殊的处理（如`immediate`立刻执行回调函数），然后借助vue响应式原理，默认在取值时会将这个旧值保存下来，然后将watcher存放到对应属性的dep中，当数据发生变化时通知对应`的watcher`重新执行`getter`函数拿到用户返回的结果作为旧值，然后执行用户传进来的回调函数，将新旧值作为参数

## 8. vue 和 react 里的key的作用是什么? 为什么不能用Index？用了会怎样? 如果不加key会怎样?

## 9. Computed 和 Methods 的区别

可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的，不同点在于computed:计算属性是基于它们的依赖进行缓存的，只有在它的相关依赖发生改变时才会重新求值而method 调用总会执行该函数。

Webpack相关面试题

一、谈谈你对webpack的理解？

答：webpack是js应用程序的静态模块打包工具，在处理应用程序的时候，内部会构建一个依赖图对应映射到项目所需要的各个模块

二、webpack的工作流程了解吗？

1. 初始化流程
2. 初始化参数：解析webpack配置参数，合并shell传入的和webpack.config.js文件配置的参数，形成最后的配置结果，各个配置项拷贝到 options 对象中
3. 初始化Compiler编译对象，为webpack声明周期不执行具体的任务，只是做调度工作
4. 加载所有配置插件plugin，监听webpack构建生命周期的事件节点做出相应的反应
5. 编译构建流程
6. 执行Compiler对象的run方法开始执行编译
7. 根据配置中的entry找出入口文件从入口文件出发开始解析文件即执行before-resolve和build module，构建AST语法树调用所有配置的Loader对模块进行编译（即执行normal-module-loader以及programe）再找出该模块依赖的模块再递归本步骤直到所有入口依赖文件都经过了处理
8. 根据入口和模块之间的依赖关系组装成一个个包含多个模块的Chunk再把每个 Chunk 转换成一个单独的文件加入到输出列表，执行seal方法并开始优化操作，在确定好输出内容后根据配置确定输出的路径和文件名把文件内容写到文件系统（即执行dist）
9. Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用Webpack 提供的 API 改变 Webpack 的运行结果
10. webpack编译构建流程即Complier -> run方法主要流程
11. compile开始编译：构建Compilation对象执行模块创建、依赖收集、分块、打包等等
12. 然后根据入口文件分析模块和依赖的模块来创建这些模块对象
13. 根据入口依赖使用工厂函数创建一下空的module对象
14. 执行buildModule，调用使用的loader完成模块的转换使用acorn解析转换后的内容输出AST，从配置的入口分析AST，当遇到require语句后将其加入到依赖的模块列表同时对新找出的这个模块再次分析
15. seal方法做了那些优化
16. 优化打包块，像提取公共文件的CommonsChunkPlugin就是在这个钩子函数触发时候进行
17. CommonsChunkPlugin提取公共代码的工作流程

CommonsChunkPlugin插件主要是为了生成webpack最后产出的asset文件