> **整理自公众号：Java专栏**
>
> 微信扫描二维码，后台回复【**导图**】，获取47张Java相关思维导图。
>
> 后台回复【**实战**】，获取5个最新微服务项目源码&&实战视屏教程（从基础环境到成功部署）

0.那你能讲一讲MVVM吗?

MVVM是`Model-View-ViewModel`缩写，也就是把`MVC`中的`Controller`演变成`ViewModel`。Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据。

1.简单说一下Vue2.x响应式数据原理

Vue在初始化数据时，会使用`Object.defineProperty`重新定义data中的所有属性，当页面使用对应属性时，首先会进行依赖收集(收集当前组件的`watcher`)如果属性发生变化会通知相关依赖进行更新操作(`发布订阅`)。

2.那你知道Vue3.x响应式数据原理吗?

(还好我有看，这个难不倒我)

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，那么Vue3又是怎样处理这个问题的呢?
>
> ”

（很简单啊）

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。

面试官抬起了头。心里暗想

(这小子还行，比上两个强，应该是多多少少看过Vue3的源码了)

3.再说一下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。这样当调用数组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。

4.nextTick知道吗，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。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。

（关于宏任务和微任务以及事件循环可以参考我的另两篇专栏）

(看到这你就会发现，其实问框架最终还是考验你的原生JavaScript功底)

5.说一下Vue的生命周期

`beforeCreate`是new Vue()之后触发的第一个钩子，在当前阶段data、methods、computed以及watch上的数据和方法都不能被访问。

`created`在实例创建完成后发生，当前阶段已经完成了数据观测，也就是可以使用数据，更改数据，在这里更改数据不会触发updated函数。可以做一些初始数据的获取，在当前阶段无法与Dom进行交互，如果非要想，可以通过vm.$nextTick来访问Dom。

`beforeMount`发生在挂载之前，在这之前template模板已导入渲染函数编译。而当前阶段虚拟Dom已经创建完成，即将开始渲染。在此时也可以对数据进行更改，不会触发updated。

`mounted`在挂载完成后发生，在当前阶段，真实的Dom挂载完毕，数据完成双向绑定，可以访问到Dom节点，使用$refs属性对Dom进行操作。

`beforeUpdate`发生在更新之前，也就是响应式数据发生更新，虚拟dom重新渲染之前被触发，你可以在当前阶段进行更改数据，不会造成重渲染。

`updated`发生在更新完成之后，当前阶段组件Dom已完成更新。要注意的是避免在此期间更改数据，因为这可能会导致无限循环的更新。

`beforeDestroy`发生在实例销毁之前，在当前阶段实例完全可以被使用，我们可以在这时进行善后收尾工作，比如清除计时器。

`destroyed`发生在实例销毁之后，这个时候只剩下了dom空壳。组件已被拆解，数据绑定被卸除，监听被移出，子实例也统统被销毁。

6.你的接口请求一般放在哪个生命周期中?

接口请求一般放在`mounted`中，但需要注意的是服务端渲染时不支持mounted，需要放到`created`中。

7.再说一下Computed和Watch

`Computed`本质是一个具备缓存的watcher，依赖的属性发生变化就会更新视图。 适用于计算比较消耗性能的计算场景。当表达式过于复杂时，在模板中放入过多逻辑会让模板难以维护，可以将复杂的逻辑放入计算属性中处理。

`Watch`没有缓存性，更多的是观察的作用，可以监听某些数据执行回调。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。这样会带来性能问题，优化的话可以使用`字符串形式`监听，如果没有写到组件中，不要忘记使用`unwatch手动注销`哦。

8.说一下v-if和v-show的区别

当条件不成立时，`v-if`不会渲染DOM元素，`v-show`操作的是样式(display)，切换当前DOM的显示和隐藏。

9.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。

10.说一下v-model的原理

`v-model`本质就是一个语法糖，可以看成是`value + input`方法的语法糖。 可以通过model属性的`prop`和`event`属性来进行自定义。原生的v-model，会根据标签的不同生成不同的事件和属性。

11.Vue事件绑定原理说一下

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。

> “
>
> 面试官：(这小子基础还可以，接下来我得上上难度了)
>
> ”

12.Vue模版编译原理知道吗，能简单说一下吗?

简单说，Vue的编译过程就是将`template`转化为`render`函数的过程。会经历以下阶段：

- 生成AST树
- 优化
- codegen

首先解析模版，生成`AST语法树`(一种用JavaScript对象的形式来描述整个模板)。 使用大量的正则表达式对模板进行解析，遇到标签、文本的时候都会执行对应的钩子进行相关处理。

Vue的数据是响应式的，但其实模板中并不是所有的数据都是响应式的。有一些数据首次渲染后就不会再变化，对应的DOM也不会变化。那么优化过程就是深度遍历AST树，按照相关条件对树节点进行标记。这些被标记的节点(静态节点)我们就可以`跳过对它们的比对`，对运行时的模板起到很大的优化作用。

编译的最后一步是`将优化后的AST树转换为可执行的代码`。

> “
>
> 面试官：(精神小伙啊，有点东西，难度提升，不信难不倒你)
>
> ”

13.Vue2.x和Vue3.x渲染器的diff算法分别说一下

简单来说，diff算法有以下过程

- 同级比较，再比较子节点
- 先判断一方有子节点一方没有子节点的情况(如果新的children没有子节点，将旧的子节点移除)
- 比较都有子节点的情况(核心diff)
- 递归比较子节点

正常Diff两个树的时间复杂度是`O(n^3)`，但实际情况下我们很少会进行`跨层级的移动DOM`，所以Vue将Diff进行了优化，从`O(n^3) -> O(n)`，只有当新旧children都为多个子节点时才需要用核心的Diff算法进行同层级比较。

Vue2的核心Diff算法采用了`双端比较`的算法，同时从新旧children的两端开始进行比较，借助key值找到可复用的节点，再进行相关操作。相比React的Diff算法，同样情况下可以减少移动节点次数，减少不必要的性能损耗，更加的优雅。

Vue3.x借鉴了 ivi算法和 inferno算法

在创建VNode时就确定其类型，以及在`mount/patch`的过程中采用`位运算`来判断一个VNode的类型，在这个基础之上再配合核心的Diff算法，使得性能上较Vue2.x有了提升。(实际的实现可以结合Vue3.x源码看。)

该算法中还运用了`动态规划`的思想求解最长递归子序列。

(看到这你还会发现，框架内无处不蕴藏着数据结构和算法的魅力)

> “
>
> 面试官：(可以可以，看来是个苗子，不过自我介绍属实有些无聊，下一题)
>
> ”

14.再说一下虚拟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`。

Vue2的Virtual DOM借鉴了开源库`snabbdom`的实现。

`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。)

VirtualDOM映射到真实DOM要经历VNode的create、diff、patch等阶段。

「key的作用是尽可能的复用 DOM 元素。」

新旧 children 中的节点只有顺序是不同的时候，最佳的操作应该是通过移动元素的位置来达到更新的目的。

需要在新旧 children 的节点中保存映射关系，以便能够在旧 children 的节点中找到可复用的节点。key也就是children中节点的唯一标识。

15.keep-alive了解吗

`keep-alive`可以实现组件缓存，当组件切换时不会对当前组件进行卸载。

常用的两个属性`include/exclude`，允许组件有条件的进行缓存。

两个生命周期`activated/deactivated`，用来得知当前组件是否处于活跃状态。

keep-alive的中还运用了`LRU(Least Recently Used)`算法。

（又是数据结构与算法，原来算法在前端也有这么多的应用）

16.Vue中组件生命周期调用顺序说一下

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`。

加载渲染过程

`父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount- >子mounted->父mounted`

子组件更新过程

`父beforeUpdate->子beforeUpdate->子updated->父updated`

父组件更新过程

`父 beforeUpdate -> 父 updated`

销毁过程

`父beforeDestroy->子beforeDestroy->子destroyed->父destroyed`

17.Vue2.x组件通信有哪些方式?

- 父子组件通信

  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`官方不推荐使用，但是写组件库时很常用

- 兄弟组件通信

  `Event Bus`实现跨组件通信`Vue.prototype.$bus = new Vue`

  `Vuex`

- 跨级组件通信

  `Vuex`

  `$attrs、$listeners`

  `Provide、inject`

18.SSR了解吗?

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有着更好的SEO、并且首屏加载速度更快等优点。不过它也有一些缺点，比如我们的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。还有就是服务器会有更大的负载需求。

19.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。

20.hash路由和history路由实现原理说一下

`location.hash`的值实际就是URL中`#`后面的东西。

history实际采用了HTML5中提供的API来实现，主要有`history.pushState()`和`history.replaceState()`。