vue面试题高薪问答攻略

内容概要

- 面试官的意图分析
- 回答策略
- 经典题目实战

面试官意图分析

烂大街的问题为什么还在问?

- 考查你的熟练度
- 考查你的深度
- 考查你的知识面
- ……

回答策略

怎么才能脱颖而出?

- 多给面试官一点
- 要有结构化的回答套路
  - 总分总
  - 3w1h
- 升华你的回答

经典面试题实战

v-if和v-for哪个优先级更高?

分析：此题考查常识，文档中曾有详细说明；也是一个很好的实践题目，项目中经常会遇到，能够看出面试者应用能力。

思路分析：总分总模式

1. 先给出结论
2. 为什么是这样的
3. 它们能放一起吗
4. 如果不能，那应该怎样
5. 总结

回答范例：

1. v-for优先于v-if被解析
2. 我曾经做过实验，把它们放在一起，输出的渲染函数中可以看出会先执行循环再判断条件
3. 实践中也不应该把它们放一起，因为哪怕我们只渲染列表中一小部分元素，也得在每次重渲染的时候遍历整个列表。
4. 通常有两种情况下导致我们这样做：
   - 为了过滤列表中的项目 (比如 `v-for="user in users" v-if="user.isActive"`)。此时定义一个计算属性 (比如 `activeUsers`)，让其返回过滤后的列表即可。
   - 为了避免渲染本应该被隐藏的列表 (比如 `v-for="user in users" v-if="shouldShowUsers"`)。此时把 `v-if` 移动至容器元素上 (比如 `ul`、`ol`)即可。
5. 文档中明确指出**永远不要把 `v-if` 和 `v-for` 同时用在同一个元素上**，显然这是一个重要的注意事项。
6. 看过源码里面关于代码生成的部分，

知其所以然：

做个测试，interview/01.html

两者同级时，渲染函数如下：

```
ƒ anonymous(
) {
with(this){return _c('div',{attrs:{"id":"app"}},_l((items),function(item)
{return (item.isActive)?_c('div',{key:item.id},[_v("\n
"+_s(item.name)+"\n    ")]):_e()}),0)}
}
```

源码中找答案compiler/codegen/index.js

你知道key的作用吗?

分析：这是一道特别常见的问题，主要考查大家对虚拟DOM和patch细节的掌握程度，能够反映面试者理解层次。

思路分析：总分总模式

1. 给出结论，key的作用是用于优化patch性能
2. key的必要性
3. 实际使用方式
4. 总结：可从源码层面描述一下vue如何判断两个节点是否相同

回答范例：

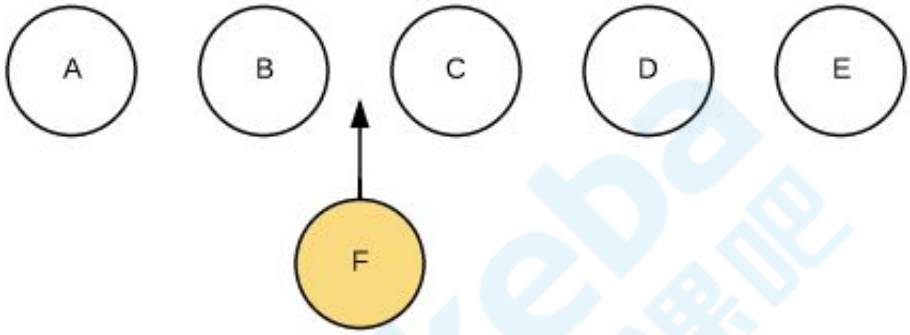
1. key的作用主要是为了更高效的更新虚拟DOM。
2. vue在patch过程中**判断两个节点是否是相同节点是key是一个必要条件**，渲染一组列表时，key往往是唯一标识，所以如果不定义key的话，vue只能认为比较的两个节点是同一个，哪怕它们实际

上不是，这导致了频繁更新元素，使得整个patch过程比较低效，影响性能。

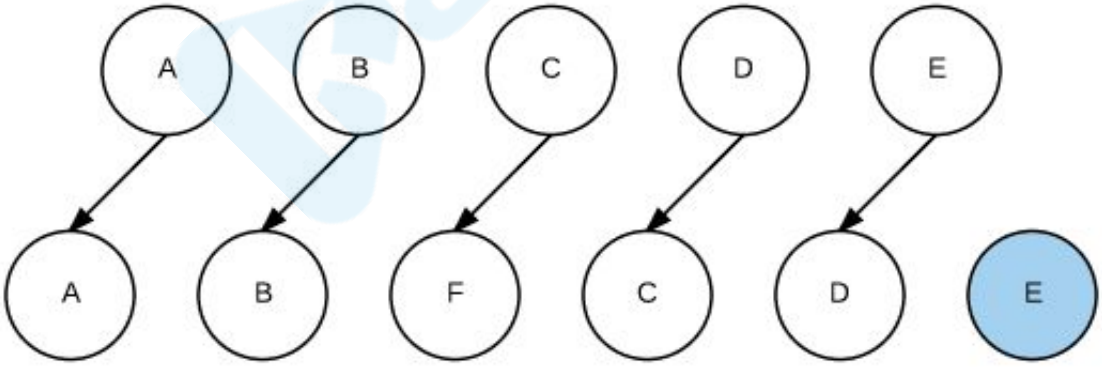
3. 实际使用中在渲染一组列表时key必须设置，而且必须是唯一标识，应该避免使用数组索引作为key，这可能导致一些隐蔽的bug；vue中在使用相同标签元素过渡切换时，也会使用key属性，其目的也是为了让vue可以区分它们，否则vue只会替换其内部属性而不会触发过渡效果。
4. 从源码中可以知道，vue判断两个节点是否相同时主要判断两者的key和元素类型等，因此如果不设置key，它的值就是undefined，则可能永远认为这是两个相同节点，只能去做更新操作，这造成了大量的dom更新操作，明显是不可取的。

测试代码，02.html

上面案例重现的是以下过程



不使用key



如果使用key

```
// 首次循环patch A
A B C D E
A B F C D E

// 第2次循环patch B
B C D E
B F C D E

// 第3次循环patch E
```

```
C D E
F C D E

// 第4次循环patch D
C D
F C D

// 第5次循环patch C
C
F C

// oldCh全部处理结束，newCh中剩下的F，创建F并插入到C前面
```

源码中找答案：src\core\vdom\patch.js - sameVnode()

能说说双向绑定以及它的实现原理吗?

题目分析：双向绑定是vue的特色之一，开发中必然会用到的知识点，然而此题还问了实现原理，升级为深度考查。

思路分析：3w1h

1. 给出双绑定义
2. 双绑带来的好处
3. 在哪使用双绑
4. 使用方式
5. 扩展：使用细节、原理实现描述

回答范例：

1. vue中双向绑定是一个指令v-model，可以绑定一个动态值到视图，同时视图中变化能改变该值。v-model是语法糖，默认情况下相当于:value和@input。
2. 使用v-model可以减少大量繁琐的事件处理代码，提高开发效率，代码可读性也更好
3. 通常在表单项上使用v-model

4. 原生的表单项可以直接使用v-model，自定义组件上如果要使用它需要在组件内绑定value并处理输入事件
5. 我做过测试，输出包含v-model模板的组件渲染函数，发现它会被转换为value属性的绑定以及一个事件监听，事件回调函数中会做相应变量更新操作，这说明神奇魔法实际上是vue的编译器完成的。

可能的追问：

1. v-model和sync修饰符有什么区别
2. 自定义组件使用v-model如果想要改变事件名或者属性名应该怎么做

知其所以然：测试代码，03.html

观察输出的渲染函数：

```
// <input type="text" v-model="foo">
// input.value = this.foo
_c('input', {
  directives: [{ name: "model", rawName: "v-model", value: (foo), expression:
"foo" }],
  attrs: { "type": "text" },
  domProps: { "value": (foo) },
  on: {
    "input": function ($event) {
      if ($event.target.composing) return;
      foo = $event.target.value
    }
  }
})
```

```
// <input type="checkbox" v-model="bar">
_c('input', {
  directives: [{ name: "model", rawName: "v-model", value: (bar), expression:
"bar" }],
  attrs: { "type": "checkbox" },
  domProps: {
    // input.checked = bar
    "checked": Array.isArray(bar) ? _i(bar, null) > -1 : (bar)
  },
  on: {
    "change": function ($event) {
      var $$a = bar, $$el = $event.target, $$c = $$el.checked ? (true) :
(false);
```

```
      if (Array.isArray($$a)) {
        var $$v = null, $$i = _i($$a, $$v);
        if ($$el.checked) { $$i < 0 && (bar = $$a.concat([$$v])) }
        else {
          $$i > -1 && (bar = $$a.slice(0, $$i).concat($$a.slice($$i + 1))) }
      } else {
        bar = $$c
      }
    }
  }
})
```

```
// <select v-model="baz">
//     <option value="vue">vue</option>
//     <option value="react">react</option>
// </select>
_c('select', {
  directives: [{ name: "model", rawName: "v-model", value: (baz), expression:
"baz" }],
  on: {
    "change": function ($event) {
      var $$selectedVal = Array.prototype.filter.call(
        $event.target.options,
        function (o) { return o.selected }
      ).map(
        function (o) {
          var val = "_value" in o ? o._value : o.value;
          return val
        }
      );
      baz = $event.target.multiple ? $$selectedVal : $$selectedVal[0]
    }
  }
}, [
  _c('option', { attrs: { "value": "vue" } }, [_v("vue")]), _v(" "),
  _c('option', { attrs: { "value": "react" } }, [_v("react")])
])
```

你了解vue中的diff算法吗?

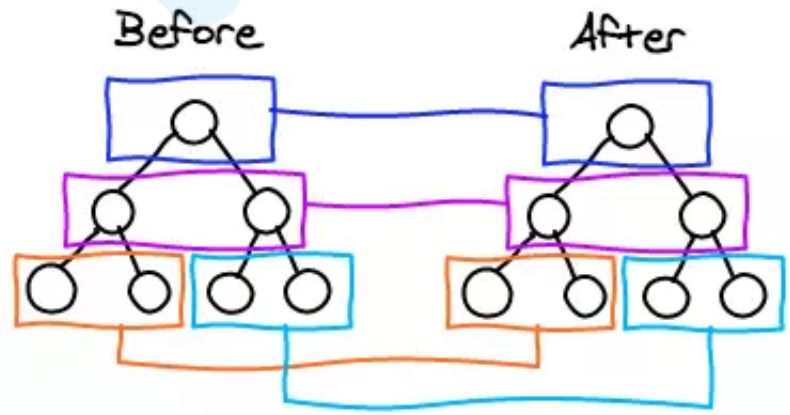
题目分析：vue基于虚拟DOM做更新，diff又是其核心部分，因此常被问道，此题考查面试者深度。

答题思路：3w1h

1. 定义diff
2. 它的必要性
3. 它在哪里被使用
4. 它如何运作
5. 提升：说一些细节

回答范例：

1. diff算法是虚拟DOM技术的产物，vue里面实际叫做patch，它的核心实现来自于snabbdom；通过新旧虚拟DOM作对比（即patch），将变化的地方转换为DOM操作
2. 在vue 1中是没有patch的，因为界面中每个依赖都有专门的watcher负责更新，这样项目规模变大就会成为性能瓶颈，vue 2中为了降低watcher粒度，每个组件只有一个watcher，但是当需要更新的时候，怎样才能精确找到发生变化的地方？这就需要引入patch才行。
3. 组件中数据发生变化时，对应的watcher会通知更新并执行其更新函数，它会执行渲染函数获取全新虚拟dom：newVnode，此时就会执行patch比对上次渲染结果oldVnode和新的渲染结果newVnode。
4. patch过程遵循深度优先、同层比较的策略；两个节点之间比较时，如果它们拥有子节点，会先比较子节点；比较两组子节点时，会假设头尾节点可能相同先做尝试，没有找到相同节点后才按照通用方式遍历查找；查找结束再按情况处理剩下的节点；借助key通常可以非常精确找到相同节点，因此整个patch过程非常高效。



知其所以然：测试代码04-patch.html

vue中组件之间的通信方式?

题目分析：vue是组件化开发框架，所以对于vue应用来说组件间的数据通信非常重要。此题主要考查大家vue基本功，对于vue基础api运用熟练度。另外一些边界知识如provide/inject/$attrs/$listeners则体现了面试者的知识面。

思路分析：总分

1. 总述知道的所有方式
2. 按组件关系阐述使用场景

回答范例：

1. 组件通信方式大体有以下8种：

- props
- $emit/$on
- $children/$parent
- $attrs/$listeners
- ref
- $root
- eventbus
- vuex

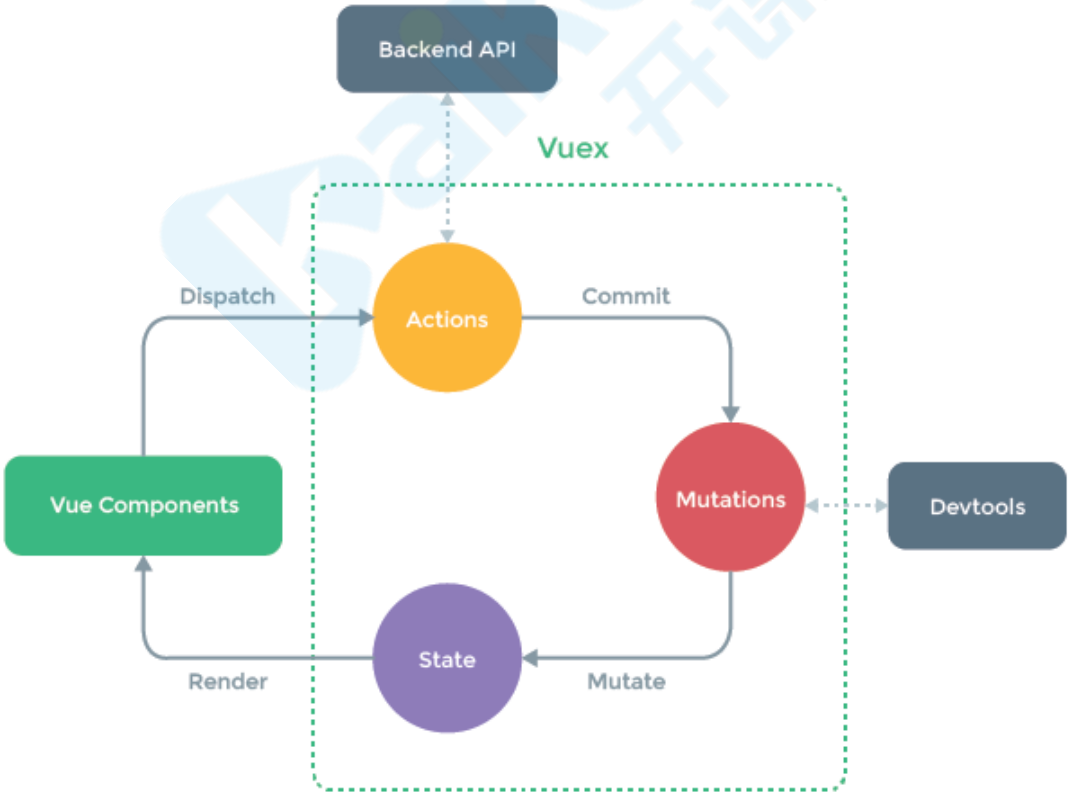
2. 根据组件之间关系讨论组件通信最为清晰有效

- 父子组件
  - `props`
  - `$emit` / `$on`
  - `$parent` / `$children`
  - `ref`
  - `$attrs` / `$listeners`
- 兄弟组件

    - `$parent`
    - `eventbus`
    - `vuex`
- 跨层级关系
    - `provide` / `inject`
    - `$root`
    - `eventbus`
    - `vuex`

简单说一说你对vuex理解?

分析：此题考查实践能力，能说出用法只能60分。更重要的是对vuex设计理念和实现原理的解读。



回答策略：3w1h

1. 首先给vuex下一个定义

2. vuex解决了哪些问题，解读理念
3. 什么时候我们需要vuex
4. 你的具体用法
5. 简述原理，提升层级

首先是官网定义：

> Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式**。它采用**集中式**存储管理应用的所有组件的状态，并以相应的规则保证状态以一种**可预测**的方式发生变化。

回答范例：

1. vuex是vue专用的状态管理库。它以全局方式集中管理应用的状态，并且可以保证状态变更的可预测性。
2. vuex主要解决的问题是多组件之间状态共享的问题，利用各种组件通信方式，我们虽然能够做到状态共享，但是往往需要在多个组件之间保持状态的一致性，这种模式很容易出现问题，也会使程序逻辑变得复杂。vuex通过把组件的共享状态抽取出来，以全局单例模式管理，这样任何组件都能用一致的方式获取和修改状态，响应式的数据也能够保证简洁的单向数据流动，我们的代码将变得更结构化且易维护。
3. vuex并非必须的，它帮我们管理共享状态，但却带来更多的概念和框架。如果我们不打算开发大型单页应用或者我们的应用并没有大量全局的状态需要维护，完全没有使用vuex的必要。一个简单的store 模式就足够了。反之，Vuex 将会成为自然而然的选择。引用 Redux 的作者 Dan Abramov 的话说就是：Flux 架构就像眼镜：您自会知道什么时候需要它。
4. 我在使用vuex过程中有如下理解：首先是对核心概念的理解和运用，将全局状态放入state对象中，它本身一棵状态树，组件中使用store实例的state访问这些状态；然后有配套的mutation方法修改这些状态，并且只能用mutation修改状态，在组件中调用commit方法提交mutation；如果应用中有异步操作或者复杂逻辑组合，我们需要编写action，执行结束如果有状态修改仍然需要提交mutation，组件中调用这些action使用dispatch方法派发。最后是模块化，通过modules选项组织拆分出去的各个子模块，在访问状态时注意添加子模块的名称，如果子模块有设置namespace，那么在提交mutation和派发action时还需要额外的命名空间前缀。
5. vuex在实现单项数据流时需要做到数据的响应式，通过源码的学习发现是借用了vue的数据响应化特性实现的，它会利用Vue将state作为data对其进行响应化处理，从而使得这些状态发生变化时，能够导致组件重新渲染。

更多题目和解答

https://github.com/57code/vue-interview

欢迎来提问，有代表性的我会更新上去，欢迎star。