12 - 怎么定义动态路由？怎么获取传过来的动态参数?

分析

API题目，考查基础能力，不容有失，尽可能说的详细。

思路

1. 什么是动态路由
2. 什么时候使用动态路由，怎么定义动态路由
3. 参数如何获取
4. 细节、注意事项

回答范例

1. 很多时候，我们需要**将给定匹配模式的路由映射到同一个组件**，这种情况就需要定义动态路由。
2. 例如，我们可能有一个 `User` 组件，它应该对所有用户进行渲染，但用户 ID 不同。在 Vue Router 中，我们可以在路径中使用一个动态字段来实现，例如：`{ path: '/users/:id', component: User }`，其中 `:id` 就是路径参数
3. *路径参数* 用冒号 `:` 表示。当一个路由被匹配时，它的 *params* 的值将在每个组件中以 `this.$route.params` 的形式暴露出来。
4. 参数还可以有多个，例如 `/users/:username/posts/:postId`；除了 `$route.params` 之外，`$route` 对象还公开了其他有用的信息，如 `$route.query`、`$route.hash` 等。

可能的追问

1. 如何响应动态路由参数的变化

https://router.vuejs.org/zh/guide/essentials/dynamic-matching.html#%E5%93%8D%E5%BA%94%E8%B7%AF%E7%94%B1%E5%8F%82%E6%95%B0%E7%9A%84%E5%8F%98%E5%8C%96

2. 我们如何处理404 Not Found路由

https://router.vuejs.org/zh/guide/essentials/dynamic-matching.html#%E6%8D%95%E8%8E%B7%E6%89%80%E6%9C%89%E8%B7%AF%E7%94%B1%E6%88%96-404-not-found-%E8%B7%AF%E7%94%B1

13-如果让你从零开始写一个vue路由，说说你的思路

思路分析：

首先思考vue路由要解决的问题：用户点击跳转链接内容切换，页面不刷新。

- 借助hash或者history api实现url跳转页面不刷新
- 同时监听hashchange事件或者popstate事件处理跳转
- 根据hash值或者state值从routes表中匹配对应component并渲染之

回答范例：

一个SPA应用的路由需要解决的问题是**页面跳转内容改变同时不刷新**，同时路由还需要以插件形式存在，所以：

1. 首先我会定义一个`createRouter`函数，返回路由器实例，实例内部做几件事：
    - 保存用户传入的配置项
    - 监听hash或者popstate事件
    - 回调里根据path匹配对应路由
2. 将router定义成一个Vue插件，即实现install方法，内部做两件事：
    - 实现两个全局组件：router-link和router-view，分别实现页面跳转和内容显示
    - 定义两个全局变量：$route和$router，组件内可以访问当前路由和路由器实例

知其所以然：

- createRouter如何创建实例

https://github1s.com/vuejs/router/blob/HEAD/src/router.ts#L355-L356

- 事件监听

https://github1s.com/vuejs/router/blob/HEAD/src/history/html5.ts#L314-L315
RouterView

- 页面跳转RouterLink

https://github1s.com/vuejs/router/blob/HEAD/src/RouterLink.ts#L184-L185

- 内容显示RouterView

https://github1s.com/vuejs/router/blob/HEAD/src/RouterView.ts#L43-L44

14-能说说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：

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

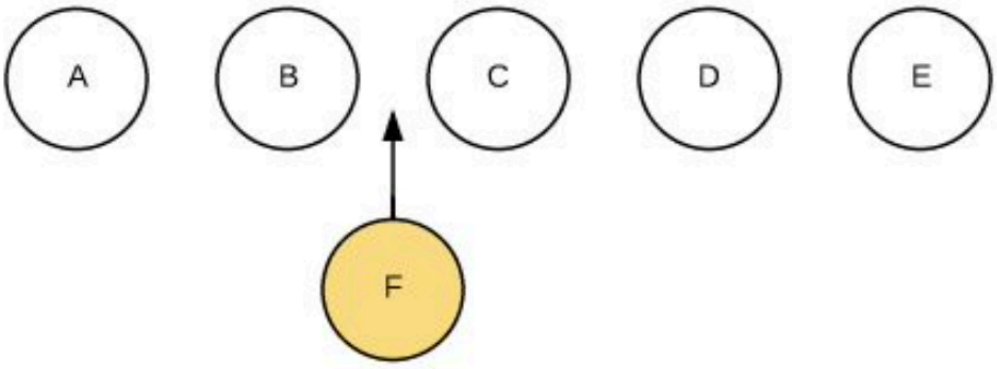
回答范例：

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。

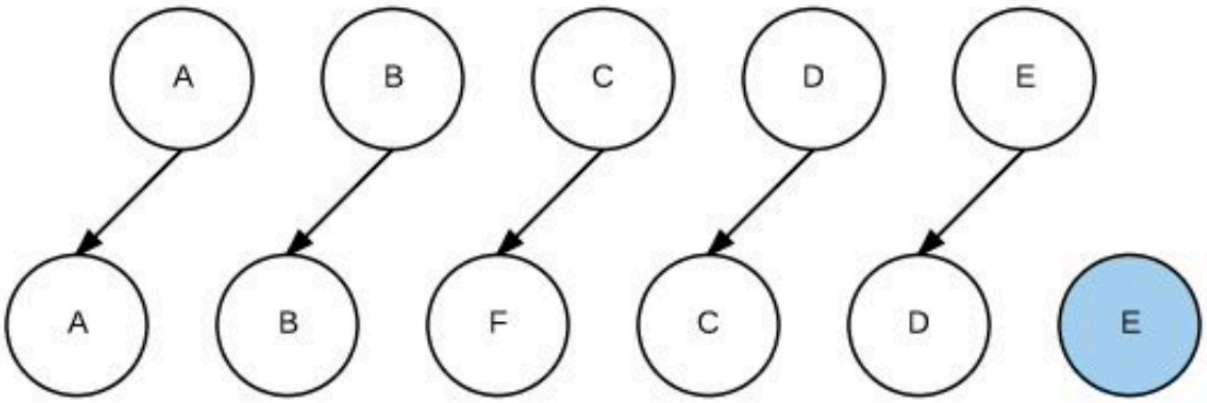
知其所以然

测试代码，[test-v3.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：

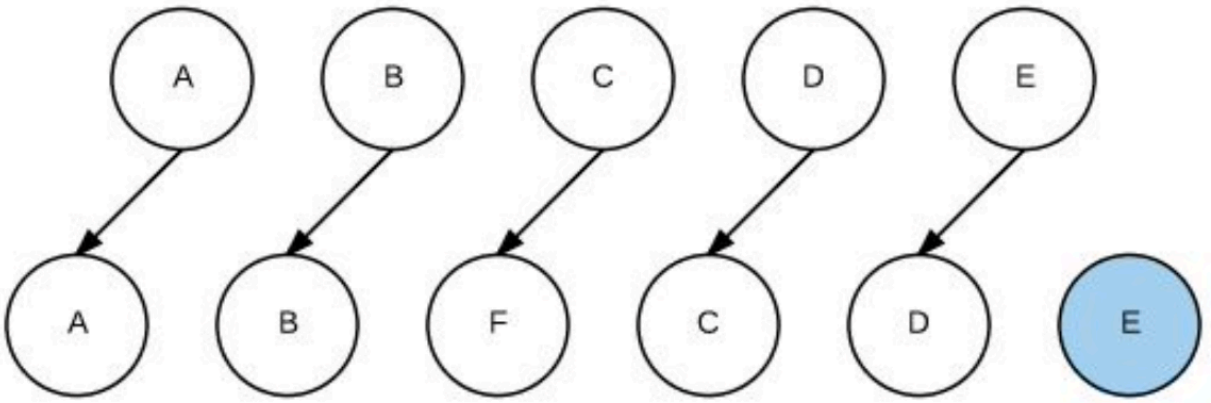
判断是否为相同节点

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/vnode.ts#L342-L343

更新时的处理

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L1752-L1753

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：

判断是否为相同节点

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/vnode.ts#L342-L343

更新时的处理

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L1752-L1753

15-说说nextTick的使用和原理?

分析

这道题及考察使用，有考察原理，nextTick在开发过程中应用的也较少，原理上和vue异步更新有密切关系，对于面试者考查很有区分度，如果能够很好回答此题，对面试效果有极大帮助。

答题思路

1. nextTick是做什么的?
2. 为什么需要它呢?
3. 开发时何时使用它？抓抓头，想想你在平时开发中使用它的地方
4. 下面介绍一下如何使用nextTick
5. 原理解读，结合异步更新和nextTick生效方式，会显得你格外优秀

回答范例：

1. nextTick是用于获取下次DOM更新刷新的使用函数。
2. Vue有个异步更新策略，意思是如果数据变化，Vue不会立刻更新DOM，而是开启一个队列，把组件更新函数保存在队列中，在同一事件循环中发生的所有数据变更会异步的批量更新。这一策略导致我们对数据的修改不会立刻体现在DOM上，此时如果想要获取更新后的DOM状态，就需要使用nextTick。
3. 开发时，比如我希望获取列表更新后的高度就可以通过nextTick实现。
4. nextTick签名如下：`function nextTick(callback?: () => void): Promise<void>`

   所以我们只需要在传入的回调函数中访问最新DOM状态即可，或者我们可以await nextTick方法返回的Promise之后做这件事。
5. 在Vue内部，nextTick之所以能够让我们看到DOM更新后的结果，是因为我们传入的callback会被添加到队列刷新函数(flushSchedulerQueue)的后面，这样等队列内部的更新函数都执行之后，所有DOM操作也就结束了，callback自然能够获取到最新的DOM值。

知其所以然：

1. 源码解读:

   更新函数入队：

   https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L1547-L1548

   入队函数：

   https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/scheduler.ts#L84-L85

   nextTick定义：

   https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/scheduler.ts#L58-L59
2. 测试案例，test-v3.html

16-watch和computed的区别以及选择?

两个重要API，反应应聘者熟练程度。

思路分析

1. 先看两者定义，列举使用上的差异
2. 列举使用场景上的差异，如何选择
3. 使用细节、注意事项
4. vue3变化

回答范例

1. 计算属性可以**从组件数据派生出新数据**，最常见的使用方式是设置一个函数，返回计算之后的结果，computed和methods的差异是它具备缓存性，如果依赖项不变时不会重新计算。侦听器**可以侦测某个响应式数据的变化并执行副作用**，常见用法是传递一个函数，执行副作用，watch没有返回值，但可以执行异步操作等复杂逻辑。
2. 计算属性常用场景是简化行内模板中的复杂表达式，模板中出现太多逻辑会是模板变得臃肿不易维护。侦听器常用场景是状态变化之后做一些额外的DOM操作或者异步操作。选择采用何用方案时首先看是否需要派生出新值，基本能用计算属性实现的方式首选计算属性。
3. 使用过程中有一些细节，比如计算属性也是可以传递对象，成为既可读又可写的计算属性。watch可以传递对象，设置deep、immediate等选项。
4. vue3中watch选项发生了一些变化，例如不再能侦测一个点操作符之外的字符串形式的表达式； reactivity API中新出现了watch、watchEffect可以完全替代目前的watch选项，且功能更加强大。

可能追问

1. watch会不会立即执行?
2. watch 和 watchEffect有什么差异

知其所以然

computed的实现

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/computed.ts#L79-L80

ComputedRefImpl

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/computed.ts#L26-L27

缓存性

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/computed.ts#L59-L60

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/computed.ts#L45-L46

watch的实现

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/apiWatch.ts#L158-L159

17-说一下 Vue 子组件和父组件创建和挂载顺序

这题考查大家对创建过程的理解程度。

思路分析

1. 给结论
2. 阐述理由

回答范例

1. 创建过程自上而下，挂载过程自下而上；即：
    - parent created
    - child created
    - child mounted
    - parent mounted
2. 之所以会这样是因为Vue创建过程是一个递归过程，先创建父组件，有子组件就会创建子组件，因此创建时现有父组件再有子组件；子组件首次创建时会添加mounted钩子到队列，等到patch结束在执行它们，可见子组件的mounted钩子是先进入到队列中的，因此等到patch结束执行这些钩子时也先执行。

知其所以然

观察beforeCreated和created钩子的处理

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/componentOptions.ts#L554-L555

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/componentOptions.ts#L741-L742

观察beforeMount和mounted钩子的处理

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L1310-L1311

测试代码，test-v3.html

18-怎么缓存当前的组件？缓存后怎么更新?

缓存组件使用keep-alive组件，这是一个非常常见且有用的优化手段，vue3中keep-alive有比较大的更新，能说的点比较多。

思路

1. 缓存用keep-alive，它的作用与用法
2. 使用细节，例如缓存指定/排除、结合router和transition
3. 组件缓存后更新可以利用activated或者beforeRouteEnter
4. 原理阐述

回答范例

1. 开发中缓存组件使用keep-alive组件，keep-alive是vue内置组件，keep-alive包裹动态组件component时，会缓存不活动的组件实例，而不是销毁它们，这样在组件切换过程中将状态保留在内存中，防止重复渲染DOM。

   ```
   <keep-alive>
     <component :is="view"></component>
   </keep-alive>
   ```

2. 结合属性include和exclude可以明确指定缓存哪些组件或排除缓存指定组件。vue3中结合vue-router时变化较大，之前是`keep-alive`包裹`router-view`，现在需要反过来用`router-view`包裹`keep-alive`：

   ```
   <router-view v-slot="{ Component }">
     <keep-alive>
       <component :is="Component"></component>
     </keep-alive>
   </router-view>
   ```

---

3. 缓存后如果要获取数据，解决方案可以有以下两种：
   - beforeRouteEnter：在有vue-router的项目，每次进入路由的时候，都会执行`beforeRouteEnter`

     ```
     beforeRouteEnter(to, from, next){
       next(vm=>{
         console.log(vm)
         // 每次进入路由执行
         vm.getData()  // 获取数据
       })
     },
     ```

   - actived：在`keep-alive`缓存的组件被激活的时候，都会执行`actived`钩子

     ```
     activated(){
         this.getData() // 获取数据
     },
     ```

---

4. keep-alive是一个通用组件，它内部定义了一个map，缓存创建过的组件实例，它返回的渲染函数内部会查找内嵌的component组件对应组件的vnode，如果该组件在map中存在就直接返回它。由于component的is属性是个响应式数据，因此只要它变化，keep-alive的render函数就会重新执行。

知其所以然

KeepAlive定义

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/components/KeepAlive.ts#L73-L74

缓存定义

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/components/KeepAlive.ts#L102-L103

缓存组件

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/components/KeepAlive.ts#L215-L216

获取缓存组件

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/components/KeepAlive.ts#L241-L242

测试缓存特性，test-v3.html

19-从0到1自己构架一个vue项目，说说有哪些步骤、哪些重要插件、目录结构你会怎么组织

综合实践类题目，考查实战能力。没有什么绝对的正确答案，把平时工作的重点有条理的描述一下即可。

思路

1. 构建项目，创建项目基本结构
2. 引入必要的插件：
3. 代码规范：prettier，eslint
4. 提交规范：husky，lint-staged
5. 其他常用：svg-loader，vueuse，nprogress
6. 常见目录结构

回答范例

1. 从0创建一个项目我大致会做以下事情：项目构建、引入必要插件、代码规范、提交规范、常用库和组件
2. 目前vue3项目我会用vite或者create-vue创建项目
3. 接下来引入必要插件：路由插件vue-router、状态管理vuex/pinia、ui库我比较喜欢element-plus和antd-vue、http工具我会选axios
4. 其他比较常用的库有vueuse，nprogress，图标可以使用vite-svg-loader
5. 下面是代码规范：结合prettier和eslint即可
6. 最后是提交规范，可以使用husky，lint-staged，commitlint

---

7. 目录结构我有如下习惯：
   `.vscode`：用来放项目中的 vscode 配置

   `plugins`：用来放 vite 插件的 plugin 配置

   `public`：用来放一些诸如 页头icon 之类的公共文件，会被打包到dist根目录下

   `src`：用来放项目代码文件

   `api`：用来放http的一些接口配置

   `assets`：用来放一些 CSS 之类的静态资源

   `components`：用来放项目通用组件

   `layout`：用来放项目的布局

   `router`：用来放项目的路由配置

   `store`：用来放状态管理Pinia的配置

   `utils`：用来放项目中的工具方法类

   `views`：用来放项目的页面文件

20-实际工作中，你总结的vue最佳实践有哪些?

看到这样的题目，可以用以下图片来回答：

思路

查看vue官方文档：

风格指南：https://vuejs.org/style-guide/

性能：https://vuejs.org/guide/best-practices/performance.html#overview

安全：https://vuejs.org/guide/best-practices/security.html

访问性：https://vuejs.org/guide/best-practices/accessibility.html

发布：https://vuejs.org/guide/best-practices/production-deployment.html

回答范例

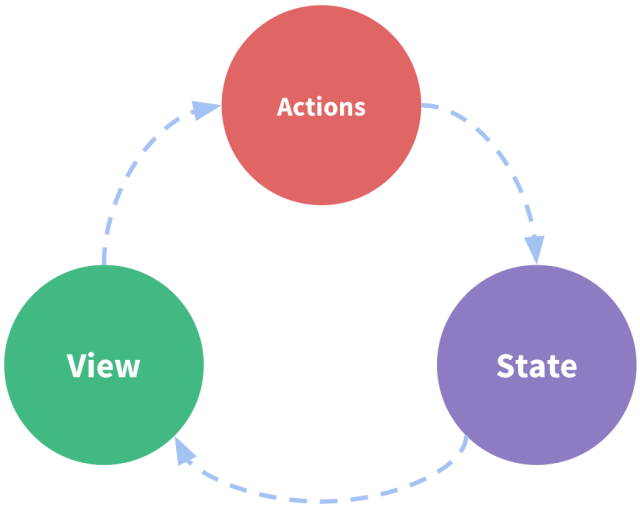
我从编码风格、性能、安全等方面说几条：

1. 编码风格方面：
    - 命名组件时使用“多词”风格避免和HTML元素冲突
    - 使用“细节化”方式定义属性而不是只有一个属性名
    - 属性名声明时使用“驼峰命名”，模板或jsx中使用“肉串命名”
    - 使用v-for时务必加上key，且不要跟v-if写在一起
2. 性能方面：
    - 路由懒加载减少应用尺寸
    - 利用SSR减少首屏加载时间
    - 利用v-once渲染那些不需要更新的内容
    - 一些长列表可以利用虚拟滚动技术避免内存过度占用
    - 对于深层嵌套对象的大数组可以使用shallowRef或shallowReactive降低开销
    - 避免不必要的组件抽象

3. 安全：
    - 不使用不可信模板，例如使用用户输入拼接模板：`template: <div> + userProvidedString + </div>`
    - 小心使用v-html，:url，:style等，避免html、url、样式等注入
4. 等等.....

---

21 - 简单说一说你对vuex理解?



思路

1. 给定义
2. 必要性阐述
3. 何时使用
4. 拓展：一些个人思考、实践经验等

范例

1. Vuex 是一个专为 Vue.js 应用开发的**状态管理模式 + 库**。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。
2. 我们期待以一种简单的“单向数据流”的方式管理应用，即状态 -> 视图 -> 操作单向循环的方式。但当我们的应用遇到**多个组件共享状态**时，比如：多个视图依赖于同一状态或者来自不同视图的行为需要变更同一状态。此时单向数据流的简洁性很容易被破坏。因此，我们有必要把组件的共享状态抽取出来，以一个全局单例模式管理。通过定义和隔离状态管理中的各种概念并通过强制规则维持视图和状态间的独立性，我们的代码将会变得更结构化且易维护。这是vuex存在的必要性，它和react生态中的redux之类是一个概念。
3. Vuex 解决状态管理的同时引入了不少概念：例如state、mutation、action等，是否需要引入还需要根据应用的实际情况衡量一下：如果不打算开发大型单页应用，使用 Vuex 反而是繁琐冗余的，一个简单的 store 模式就足够了。但是，如果要构建一个中大型单页应用，Vuex 基本是标配。

4. 我在使用vuex过程中感受到一些blabla

可能的追问

1. vuex有什么缺点吗？你在开发过程中有遇到什么问题吗?
2. action和mutation的区别是什么？为什么要区分它们?

22-说说从 template 到 render 处理过程

分析

问我们template到render过程，其实是问vue `编译器` 工作原理。

思路

1. 引入vue编译器概念
2. 说明编译器的必要性
3. 阐述编译器工作流程

回答范例

1. Vue中有个独特的编译器模块，称为“compiler”，它的主要作用是将用户编写的template编译为js中可执行的render函数。
2. 之所以需要这个编译过程是为了便于前端程序员能高效的编写视图模板。相比而言，我们还是更愿意用HTML来编写视图，直观且高效。手写render函数不仅效率底下，而且失去了编译期的优化能力。
3. 在Vue中编译器会先对template进行解析，这一步称为parse，结束之后会得到一个JS对象，我们成为抽象语法树AST，然后是对AST进行深加工的转换过程，这一步成为transform，最后将前面得到的AST生成为JS代码，也就是render函数。

知其所以然

vue3编译过程窥探：

https://github1s.com/vuejs/core/blob/HEAD/packages/compiler-core/src/compile.ts#L61-L62

测试，test-v3.html

可能的追问

1. Vue中编译器何时执行?
2. react有没有编译器?

23-Vue实例挂载的过程中发生了什么?

分析

挂载过程完成了最重要的两件事：

1. 初始化
2. 建立更新机制

把这两件事说清楚即可！

回答范例

1. 挂载过程指的是app.mount()过程，这个过程中整体上做了两件事：**初始化**和**建立更新机制**
2. 初始化会创建组件实例、初始化组件状态，创建各种响应式数据
3. 建立更新机制这一步会立即执行一次组件更新函数，这会首次执行组件渲染函数并执行patch将前面获得vnode转换为dom；同时首次执行渲染函数会创建它内部响应式数据之间和组件更新函数之间的依赖关系，这使得以后数据变化时会执行对应的更新函数。

知其所以然

测试代码，test-v3.html

mount函数定义

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/apiCreateApp.ts#L277-L278

首次render过程

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L2303-L2304

可能的追问

1. 响应式数据怎么创建
2. 依赖关系如何建立