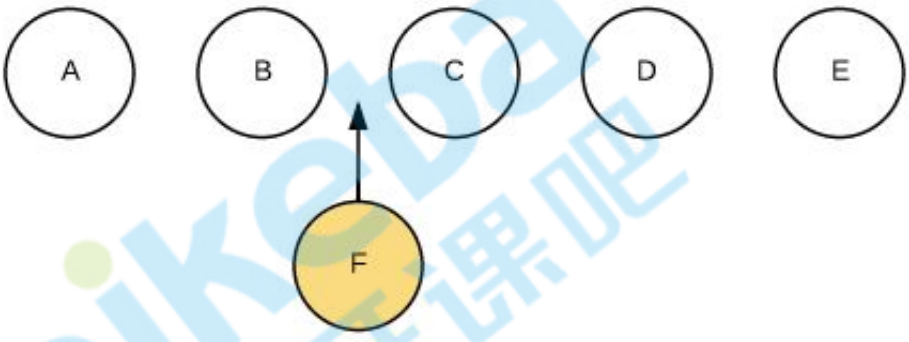
第四阶段 Vue面试题解析

你知道vue中key的作用和工作原理吗？说说你对它的理解。

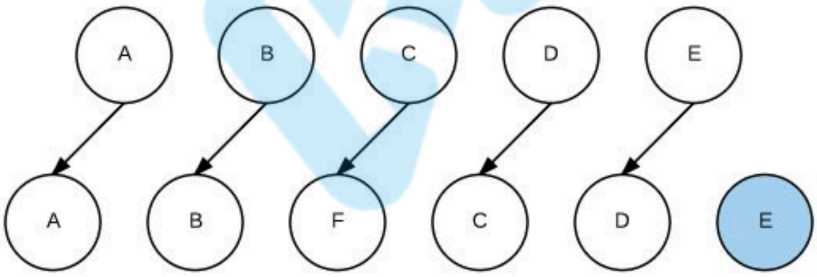
测试代码如下

```
<!DOCTYPE html>
<html>

<head>
    <title>key的作用及原理?</title>
</head>

<body>
    <div id="demo">
        <p v-for="item in items" :key="item">{{item}}</p>
    </div>
    <script src="https://unpkg.com/vue"></script>
    <script>
        // 创建实例
        const app = new Vue({
            el: '#demo',
            data: { items: ['a', 'b', 'c', 'd', 'e'] },
            mounted () {
                setTimeout(() => {
                    this.items.splice(2, 0, 'f')
                }, 2000);
            },
        });
    </script>
</body>

</html>
```

上面案例更新过程重现



不使用key



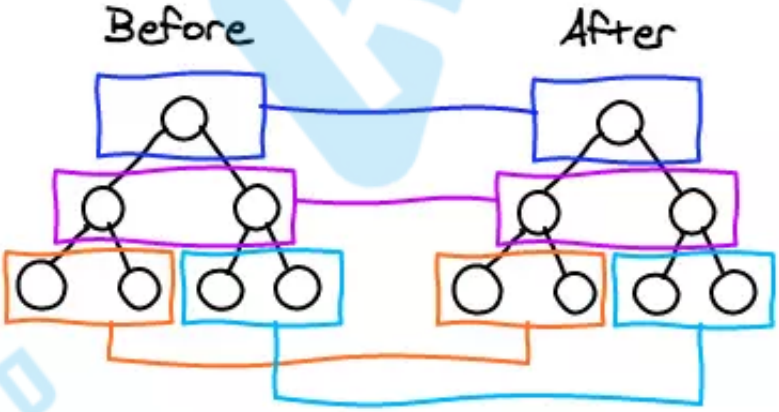
如果使用key

```
// 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
```

结论

1. key的作用是vue在patch过程中通过key可以判断两个节点是否是同一个，从而实现复用，避免频繁更新不同元素，使得整个更新过程更高效，减少DOM操作量，提高性能。
2. 你是怎么知道的
3. 实践中key设置方式
4. 实践中，如果相同标签名元素要做transition切换时，也会用到key，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。

你怎么理解vue中的diff算法?



总结

1.概念：vue中的diff算法叫patch，发生在程序更新阶段。目标是通过新旧虚拟DOM的对比，将变化的地方更新在真实DOM上。

2.必要性：vue 2.x中降低了Watcher粒度，每个组件会有一个渲染Watcher与之对应，只有引入diff才能精确找到组件中发生变化的地方。

3.执行过程：diff过程整体遵循**深度优先、同层比较**的策略；两个节点之间比较时，先做属性更新，然后会根据它们是否拥有子节点或者文本节点的情况做不同操作；比较两组子节点是算法的重点，首先假设头尾节点可能相同做4次比对尝试，找到直接更新，没找到则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，老数组剩余批量删除，新数组剩余批量创建。

Vue3.0做了哪些优化?

Vue3.0改进主要在以下几点：

- 更快
  - 编译阶段优化：静态提升、动态属性标记、区块
  - 基于Proxy的响应式系统
- 更小：通过摇树优化核心库体积
- 更容易维护：TypeScript + 模块化
- 更加友好
  - 跨平台：编译器核心和运行时核心与平台无关，使得Vue更容易与任何平台（Web、Android、iOS）一起使用
- 更容易使用
  - 改进的TypeScript支持，编辑器能提供强有力的类型检查和错误及警告
  - 更好的调试支持
  - 独立的响应化模块
  - Composition API

watch和computed的区别以及怎么选用?

先看用法:

```
{
  data() {
    return {
      counter: 1
    }
  },
  watch: {
    counter(newValue, oldValue) {
      // do something
    }
  },
  computed: {
    double() {
      return this.counter * 2
    }
  },
}
```

总结:

概念上的区别:

- watch:侦听响应式数据的变化,若数据变化执行副作用函数
- computed:由响应式数据派生出的新数据

用法上的区别:

- watch:需要明确指定侦听目标,可以获取目标变化前后的值,不需要返回值。
- computed:不能指定侦听目标,不能获取目标变化前后的值,需要返回值。

如何选择:

- 选watch的情况:
  - 需要指定确切侦听目标,或需要变化前后的值;
  - 响应数据变化时需要做异步操作或耗时操作时。
- 大部分情况可选computed

你知道vue双向数据绑定的原理吗?

测试代码

```
<div id="app">
  <input v-model="counter" type="text" />
</div>
<script src="https://unpkg.com/vue"></script>
<script>
  const app = new Vue({
    data() {
      return {
        counter: 1
      }
    },
  }).$mount('#app')

  console.log(app.$options.render);
</script>
```

渲染函数

```
(function anonymous(
) {
with(this){return _c('div',{attrs:{"id":"app"}},[
 _c('input',{directives:[{
   name:"model",rawName:"v-model",value:(counter),expression:"counter"}],
   attrs:{"type":"text"},
   domProps:{"value":(counter)},
   on:{"input":function($event){
     if($event.target.composing)
       return;counter=$event.target.value}}})])}
})
```

结论:

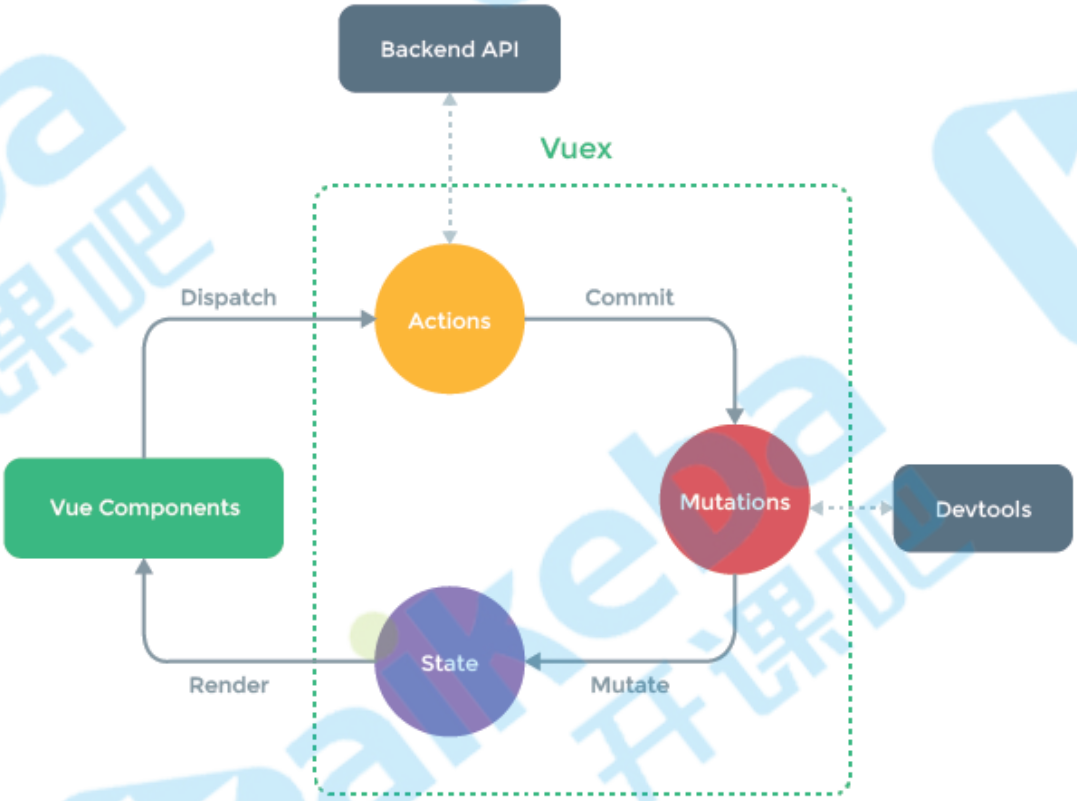
1. 双向绑定是个语法糖
2. 它将`v-model="counter"`指令转换为dom属性value的赋值和事件input的监听
3. 说出你怎么知道的

可能的追问:

1. 自定义组件也这样吗?
2. vue3中双绑有什么变化

简单说一说vuex使用及其理解?

此题考查基本能力，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：

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。Vuex 也集成到 Vue 的官方调试工具 devtools extension，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

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。

vue中组件之间的通信方式?

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则提现了面试者的广度。

下面是一个组件通信方式总结，共8种：

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

根据组件之间关系讨论组件通信最为清晰有效

- 父子组件之间通信
  - `props`
  - `$emit` / `$on`
  - `$parent` / `$children`
  - `ref`
  - `$attrs` / `$listeners`
- 兄弟组件
  - `$parent`
  - `$root`
  - `eventbus`
  - `vuex`
- 跨层级关系
  - `eventbus`
  - `vuex`
  - `provide` / `inject`

vue-router中如何保护指定路由的安全?

此题是考查项目实践能力，项目中基本都有路由守卫的需求，保护指定路由考查的就是这个知识点。

答题整体思路：

1. 阐述vue-router中路由保护策略
2. 描述具体实现方式
3. 简单说一下它们是怎么生效的

回答范例：

1. vue-router中保护路由安全通常使用导航守卫来做，通过设置路由导航钩子函数的方式添加守卫函数，在里面判断用户的登录状态和权限，从而达到保护指定路由的目的。
2. 具体实现有几个层级：全局前置守卫beforeEach、路由独享守卫beforeEnter或组件内守卫beforeRouteEnter。以全局守卫为例来说，可以使用`router.beforeEach((to,from,next)=>{})`方式设置守卫，每次路由导航时，都会执行该守卫，从而检查当前用户是否可以继续导航，通过给next函数传递多种参数达到不同的目的，比如如果禁止用户继续导航可以传递next(false)，正常放行可以不传递参数，传递path字符串可以重定向到一个新的地址等等。
3. 这些钩子函数之所以能够生效，也和vue-router工作方式有关，像beforeEach只是注册一个hook，当路由发生变化，router准备导航之前会批量执行这些hooks，并且把目标路由to，当前路由from，以及后续处理函数next传递给我们设置的hook。

可能的追问：

1. 能不能说说全局守卫、路由独享守卫和组件内守卫区别?
    - 作用范围
    - 组件实例的获取

```
beforeRouteEnter(to,from,next) {
  next(vm => {

  })
}
```

    - 名称/数量/顺序

> 1. 导航被触发。
> 2. 在失活的组件里调用离开守卫。
> 3. 调用全局的 `beforeEach` 守卫。
> 4. 在重用的组件里调用 `beforeRouteUpdate` 守卫 (2.2+)。
> 5. 在路由配置里调用 `beforeEnter`。
> 6. 解析异步路由组件。
> 7. 在被激活的组件里调用 `beforeRouteEnter`。
> 8. 调用全局的 `beforeResolve` 守卫 (2.5+)。
> 9. 导航被确认。
> 10. 调用全局的 `afterEach` 钩子。
> 11. 触发 DOM 更新。
> 12. 用创建好的实例调用 `beforeRouteEnter` 守卫中传给 `next` 的回调函数。

2. 你项目中的路由守卫是怎么做的?
3. 前后端路由一样吗?
4. 前端路由是用什么方式实现的?
5. 你前面提到的next方法是怎么实现的?

你知道nextTick吗，它是干什么的，实现原理是什么?

这道题考查大家对vue异步更新队列的理解，有一定深度，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是啥？下一个定义
2. 为什么需要它呢？用异步更新队列实现原理解释
3. 我再什么地方用它呢？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. 最后能说出源码实现就会显得你格外优秀

先看看官方定义

> Vue.nextTick([callback, context])
>
> 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这个方法，获取更新后的 DOM。

```
// 修改数据
vm.msg = 'Hello'
// DOM 还没有更新
Vue.nextTick(function () {
// DOM 更新了
})
```

回答范例：

1. nextTick是Vue提供的一个全局API，由于vue的异步更新策略导致我们对数据的修改不会立刻体现在dom变化上，此时如果想要立即获取更新后的dom状态，就需要使用这个方法
2. Vue 在更新 DOM 时是**异步**执行的。只要侦听到数据变化，Vue 将开启一个队列，并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，只会被推入到队列中一次。这种在缓冲时去除重复数据对于避免不必要的计算和 DOM 操作是非常重要的。nextTick方法会在队列中加入一个回调函数，确保该函数在前面的dom操作完成后才调用。
3. 所以当我们想在修改数据后立即看到dom执行结果就需要用到nextTick方法。
4. 比如，我在干什么的时候就会使用nextTick，传一个回调函数进去，在里面执行dom操作即可。
5. 我也有简单了解nextTick实现，它会在callbacks里面加入我们传入的函数，然后用timerFunc异步方式调用它们，首选的异步方式会是Promise。这让我明白了为什么可以在nextTick中看到dom操作结果。

说一说你对vue响应式理解?

烂大街的问题，但却不是每个人都能回答到位。因为如果你只是看看别人写的网文，通常没什么底气，也经不住面试官推敲，但像我们这样即看过源码还造过轮子的，回答这个问题就会比较有底气。

答题思路：

1. 啥是响应式?
2. 为什么vue需要响应式?
3. 它能给我们带来什么好处?
4. vue的响应式是怎么实现的？有哪些优缺点?
5. vue3中的响应式的新变化

回答范例：

1. 所谓数据响应式就是能够使数据变化可以被检测并对这种变化做出响应的机制。
2. mvvm框架中要解决的一个核心问题是连接数据层和视图层，通过数据驱动应用，数据变化，视图更新，要做到这点的就需要对数据做响应式处理，这样一旦数据发生变化就可以立即做出更新处理。
3. 以vue为例说明，通过数据响应式加上虚拟DOM和patch算法，可以使我们只需要操作数据，完全不用接触繁琐的dom操作，从而大大提升开发效率，降低开发难度。
4. vue2中的数据响应式会根据数据类型来做不同处理，如果是对象则采用Object.defineProperty()的方式定义数据拦截，当数据被访问或发生变化时，我们感知并作出响应；如果是数组则通过覆盖该数组原型的方法，扩展它的7个变更方法，使这些方法可以额外的做更新通知，从而作出响应。这种机制很好的解决了数据响应化的问题，但在实际使用中也存在一些缺点：比如初始化时的递归遍历会造成性能损失；新增或删除属性时需要用户使用Vue.set/delete这样特殊的api才能生效；对于es6中新产生的Map、Set这些数据结构不支持等问题。
5. 为了解决这些问题，vue3重新编写了这一部分的实现：利用ES6的Proxy机制代理要响应化的数据，它有很多好处，编程体验是一致的，不需要使用特殊api，初始化性能和内存消耗都得到了大幅改善；另外由于响应化的实现代码抽取为独立的reactivity包，使得我们可以更灵活的使用它，我们甚至不需要引入vue都可以体验。

你如果想要扩展某个Vue组件时会怎么做?

此题属于实践题，着重考察大家对vue常用api使用熟练度，答题时不仅要列出这些解决方案，同时最好说出他们异同。

答题思路：

按照逻辑扩展和内容扩展来列举，逻辑扩展有：mixins、extends、composition api；内容扩展有slots；

分别说出他们使用使用方法、场景差异和问题。

作为扩展，还可以说说vue3中新引入的composition api带来的变化

回答范例：

1. 常见的组件扩展方法有：mixins，slots，extends等
2. 混入mixins是分发 Vue 组件中可复用功能的非常灵活的方式。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

```
// 复用代码：它是一个配置对象，选项和组件里面一样
const mymixin = {
  methods: {
    dosomething(){}
  }
}
// 全局混入：将混入对象传入
Vue.mixin(mymixin)

// 局部混入：做数组项设置到mixins选项，仅作用于当前组件
const Comp = {
  mixins: [mymixin]
}
```

3. 插槽主要用于vue组件中的内容分发，也可以用于组件扩展。

   子组件Child

```
<div>
  <slot>这个内容会被父组件传递的内容替换</slot>
</div>
```

   父组件Parent

```
<div>
  <Child>来自老爹的内容</Child>
</div>
```

   如果要精确分发到不同位置可以使用具名插槽，如果要使用子组件中的数据可以使用作用域插槽。

4. 组件选项中还有一个不太常用的选项extends，也可以起到扩展组件的目的

```
// 扩展对象
const myextends = {
  methods: {
    dosomething(){}
  }
}
// 组件扩展：做数组项设置到extends选项，仅作用于当前组件
// 跟混入的不同是它只能扩展单个对象
// 另外如果和混入发生冲突，该选项优先级较高，优先起作用
const Comp = {
  extends: myextends
}
```

5. 混入的数据和方法不能明确判断来源且可能和当前组件内变量产生命名冲突，vue3中引入的composition api，可以很好解决这些问题，利用独立出来的响应式模块可以很方便的编写独立逻辑并提供响应式的数据，然后在setup选项中有机组合使用。例如：

```
// 复用逻辑1
function useXX() {}
// 复用逻辑2
function useYY() {}
// 逻辑组合
const Comp = {
  setup() {
    const {xx} = useXX()
    const {yy} = useYY()
    return {xx, yy}
  }
}
```

可能的追问

Vue.extend方法你用过吗？它能用来做组件扩展吗？