1. 为何Vue采用异步渲染。

防止数据一改变就重新渲染，数据改变的多会渲染频繁。

异步渲染实现：根据watcher的id进行过滤，判断是否放入到queueWatcher中，然后通过nextTick来异步刷新queueWatcher（执行watcher）。

1. nextTick 实现原理。

将nextTick要执行的回调函数通过一个匿名函数包裹放入到callbacks数组中，然后使用promise和settimeout异步依次callbacks中方法。

1. Computed特点。

具备缓存，只有当依赖的属性发生变化才会重新计算。默认dirty属性为true，如果dirty为true则会求值，求完值dirty变为false，dirty为false时候返回上次计算结果，如果依赖属性改变则dirty变为true（在watcher的update方法中）

1. Watch中 deep:true是如何实现的。

如果是deep，会递归监听

1. beforeDestory使用场景

解绑$on,清除定时器，接触事件绑定scroll，mousemove

1. Vue模板编译原理  
   将模板->ast（抽象语法树）->render函数
2. V-for和v-if为何不能连用

v-for优先级比v-if高，先循环再条件判断，性能低，用template包裹，v-if定义在template上。

8. diff算法

1.先同级比较，再比较子节点。

2.新的有子节点，老的没有，新的插到老的。新的没有，老的有。将老的删除。

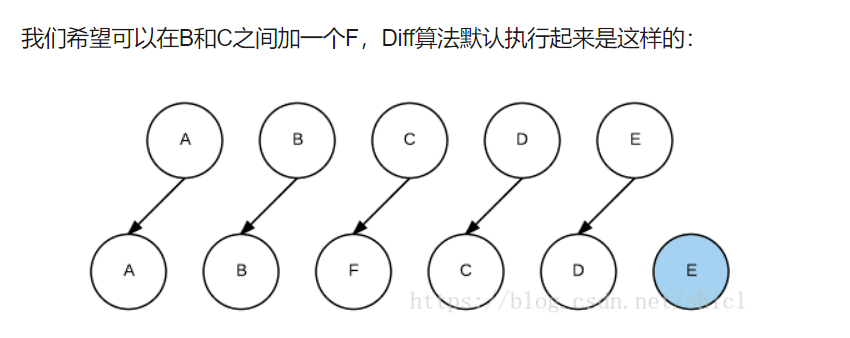
3.比较都有子节点的情况。

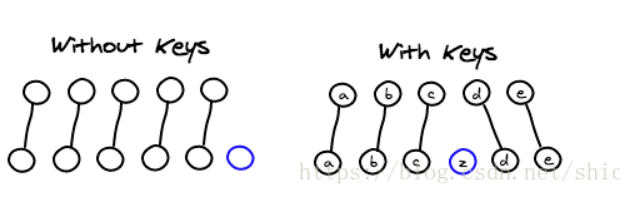
4.递归比较子节点

9. v-for中为什么使用key

不用key它会复用相同标签

高效的更新虚拟DOM

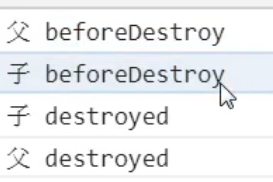
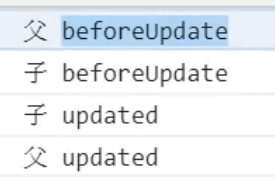
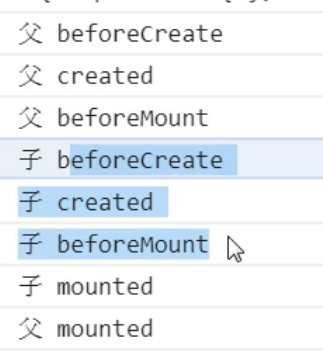




10. 描述组件渲染和更新过程

渲染组件时，会通过Vue.extend方法构建子组件构造函数，并进行实例化，最终手动调用$mount()进行挂载

11.父子组件生命周期调用顺序



12．组件通信，单相数据流

父子间通信 父->子通过props 子->$emit(发布订阅)。

$parent,$root,$children获取组件实例。

Ref获取实例

Provide，inject注入。

Bus

Vuex

13.为什么使用异步组件（懒加载）

防止打包出来的的结果太大

14.响应式原理

在initData()初始化数据中在通过proxy使this.age能访问到this.\_data.age。然后通过observe观测data中的数据，如果不是对象则返回，如果是数组则对数组中的每一项调用observe观测。

如果是对象则调用defineReactice对其响应式化，先实例化一个dep，如果对象中属性的值依旧是对象则递归调用observe。通过使用Object.defineProperty对于对象的每一个key设置get，set方法，在get方法中调用dep.depend进行依赖收集，在依赖收集中，会将dep对象添加到当前watcher的deps数组中，并将当前watcher添加到dep的subs数组中。当修改数据时候会触发set函数，通过dep.notify()来派发更新。通过updateComponent重新渲染视图。