**1. 请简述 React 16 版本中初始渲染的流程**

要将 React 元素渲染到页面中，分为两个阶段，render 阶段和 commit 阶段。

render 阶段负责创建 Fiber 数据结构并为 Fiber 节点打标记，标记当前 Fiber 节点要进行的 DOM 操作。

commit 阶段负责根据 Fiber 节点标记 ( effectTag ) 进行相应的 DOM 操作。

**2. 为什么 React 16 版本中 render 阶段放弃了使用递归**

React 16 之前的组件渲染方式是递归渲染：渲染父节点 -> 渲染子节点

递归渲染看起来十分简单，但是如果想在子节点的渲染过程中执行优先级更高的操作，只能保留调用栈中子节点的渲染及子节点之前节点的渲染，这样是很复杂的，这种调和/渲染也叫做 Stack Reconciler。

**Fiber Reconciler**

Fiber 使用链表的结构去渲染节点，每一个节点都称之为 Fiber Node，每个节点会有三个属性：

* child 指向第一个子节点
* sibling 指向兄弟节点
* return 指向父节点

Fiber 的渲染方式：从父节点开始，向下依次遍历子节点，深度优先渲染完子节点后，再回到其父节点去检查是否有兄弟节点，如果有兄弟节点，则从该兄弟节点开始继续深度优先的渲染，直到回退到根节点结束。

**重复遍历的节点并不会重复渲染，而是为了取到下一个可能需要渲染的节点。**

此时每一个节点都是一个渲染任务, 从而将整个界面渲染任务拆分成更小的模块，渲染可拆分就意味着每次任务执行前都可以检查是否去执行优先级更高的操作。

**3. 请简述 React 16 版本中 commit 阶段的三个子阶段分别做了什么事情**

commit 阶段可以分为三个子阶段：

* before mutation 阶段（执行 DOM 操作前）
* mutation 阶段（执行 DOM 操作）
* layout 阶段（执行 DOM 操作后）

4. 请简述 workInProgress Fiber 树存在的意义是什么

React 在 render 第一次渲染时，会通过 React.createElement 创建一颗 Element 树，可以称之为 **Virtual DOM Tree**，由于要记录上下文信息，加入了 Fiber，每一个 Element 会对应一个 Fiber Node，将 Fiber Node 链接起来的结构成为 **Fiber Tree**。它反映了用于渲染 UI 的应用程序的状态。这棵树通常被称为 **current 树（当前树，记录当前页面的状态）。**

在后续的更新过程中（setState），每次重新渲染都会重新创建 Element, 但是 Fiber 不会，Fiber 只会使用对应的 Element 中的数据来更新自己必要的属性，

Fiber Tree 一个重要的特点是链表结构，将递归遍历编程循环遍历，然后配合 requestIdleCallback API, 实现任务拆分、中断与恢复。

每一个 Fiber Node 节点与 Virtual Dom 一一对应，所有 Fiber Node 连接起来形成 Fiber tree, 是个单链表树结构。

**当 render 的时候有了这么一条单链表，当调用 setState 的时候又是如何 Diff 得到 change 的呢？**

采用的是一种叫**双缓冲技术（double buffering）**，这个时候就需要另外一颗树：WorkInProgress Tree，它反映了要刷新到屏幕的未来状态。

WorkInProgress Tree 构造完毕，得到的就是新的 Fiber Tree，然后喜新厌旧（把 current 指针指向WorkInProgress Tree，丢掉旧的 Fiber Tree）就好了。

这样做的好处：

* 能够复用内部对象（fiber）
* 节省内存分配、GC的时间开销
* 就算运行中有错误，也不会影响 View 上的数据

每个 Fiber上都有个alternate属性，也指向一个 Fiber，创建 WorkInProgress 节点时优先取alternate，没有的话就创建一个。

创建 WorkInProgress Tree 的过程也是一个 Diff 的过程，Diff 完成之后会生成一个 Effect List，这个 Effect List 就是最终 Commit 阶段用来处理副作用的阶段。