React15架构（同步更新）	>React16架构(可中断异步更新)
Stack reconciler（协调器） 负责找出变化的组件	**Scheduler（调度器）** 调度任务的优先级，高优任务优先进入Reconciler
↓	↓
Renderer（渲染器） 负责将变化的组件渲染到页面上	**Fiber reconciler（协调器）** 负责找出变化的组件
	↓
	Renderer（渲染器） 负责将变化的组件渲染到页面上

1

Stack reconciler：
1、调用函数组件、或class组件的render方法，将返回的JSX转化为虚拟DOM
2、将虚拟DOM和上次更新时的虚拟DOM对比
3、通过对比找出本次更新中变化的虚拟DOM
4、通知Renderer将变化的虚拟DOM渲染到页面上

Stack reconciler实现说明：https://zh-hans.reactjs.org/docs/implementation-notes.html

同步更新

缺点：
由于递归更新，无法中断，更新时间超过16ms，出现卡顿

Reconciler和Renderer是交替工作：

2

Fiber reconciler：
1、支持任务不同优先级，可中断与恢复，并且恢复后可以复用之前的中间状态
2、任务更新单元为React Element对应的Fiber节点
3、没有采用Generator实现协调器，自己实现了一套异步可中断更新机制
4、Fiber 取代 React16虚拟DOM的叫法
5、基于Fiber节点实现，被称为Fiber Reconciler

3

异步可中断更新

> JS脚本执行 ----- 样式布局 ----- 样式绘制
在此过程中，react预留5ms给JS进程，超过5ms会将控制权还于渲染进程，中断更新，下一帧再调起

> 优先级更高的更新任务被调度，导致中断

React16架构中整个更新流程为(红框中的工作都在内存中进行)：

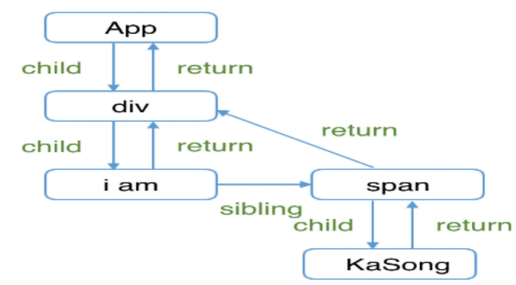
Renderer根据Reconciler为虚拟DOM打的标记，同步执行对应的DOM操作。

Fiber节点组成
静态的数据结构：每个Fiber节点对应一个React element，保存了该组件的类型（函数组件/类组件/原生组件...）、对应的DOM节点等信息。
连接其他Fiber节点形成Fiber树的属性
作为动态的工作单元的属性：每个Fiber节点保存了本次更新中该组件改变的状态、要执行的工作（需要被删除/被插入页面中/被更新...）
调度优先级相关的属性
指向该fiber在另一次更新时对应的fiber属性

4

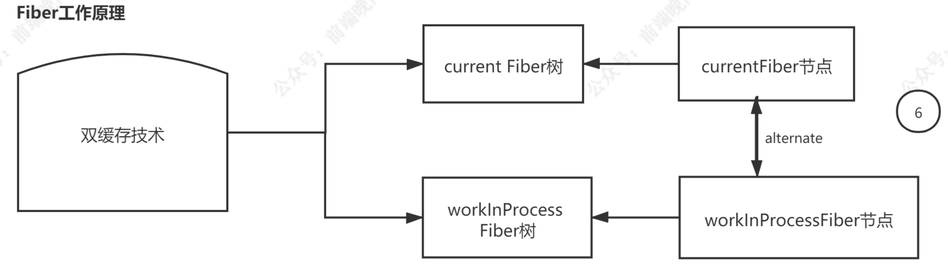
```
function FiberNode(
  tag: WorkTag,
  pendingProps: mixed,
  key: null | string,
  mode: TypeOfMode,
) {

  /***** 作为静态数据结构的属性 *****/
  this.tag = tag;
  this.key = key;
  this.elementType = null;
  this.type = null;
  this.stateNode = null;

  /***** 用于连接其他Fiber节点形成Fiber树 *****/
  this.return = null;  // 指向父级Fiber节点，return指节点执行完completeWork后会返回的下一个节点
  this.child = null; //  指向子Fiber节点
  this.sibling = null; // 指向右边第一个兄弟Fiber节点
  this.index = 0;

  this.ref = null;

  /***** 作为动态的工作单元的属性 *****/
  // 保存本次更新造成的状态改变相关信息
  this.pendingProps = pendingProps;
  this.memoizedProps = null;
  this.updateQueue = null;
  this.memoizedState = null;
  this.dependencies = null;

  this.mode = mode;

  // 保存本次更新会造成的DOM操作
  this.effectTag = NoEffect;
  this.nextEffect = null;

  this.firstEffect = null;
  this.lastEffect = null;

  /***** 调度优先级相关 *****/
  this.lanes = NoLanes;
  this.childLanes = NoLanes;

  /***** 指向该fiber在另一次更新时对应的fiber *****/
  this.alternate = null;
}
```

Fiber树的结构（深度优先遍历）



5

Fiber工作原理



6

current Fiber树：当前屏幕上显示内容对应的Fiber树称为current Fiber树

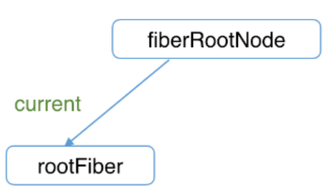
workInProgress Fiber树： 正在内存中构建的Fiber树称为workInProgress Fiber树

fiberRootNode与rootFiber：fiberRootNode是整个应用的根节点（挂载元素），rootFiber：根组件（如<App />）所在组件树根节点

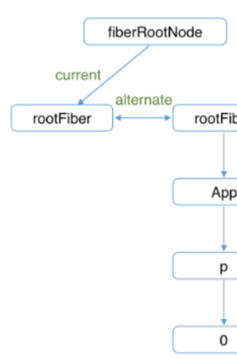
fiberRootNode.current = rootFiber，当workInProgress Fiber树构建完成交给Renderer渲染在页面上后，应用根节点的current指针指向workInProgress Fiber树，此时workInProgress Fiber树就变为current Fiber树，这样来完成Dom更新

Fiber Mount与Update流程概览

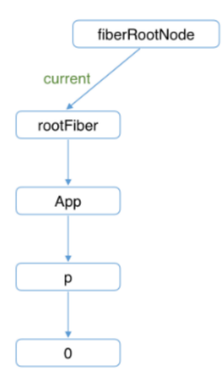
Mount时



首屏渲染，current树为空，只要rootFiber节点

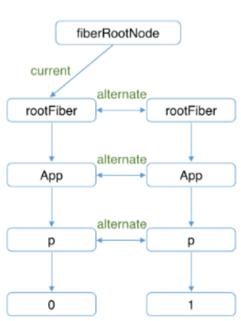


render阶段，JSX在内存依次创建Fiber节点，构建workInProcess Fiber树，尝试复用current rootFiber

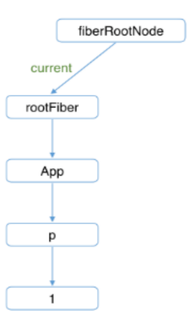


commit阶段，workInProcess Fiber树渲染成页面，改变fiberRoot current指针指向，workInProcess Fiber树变为current Fiber树，完成mount

Update时



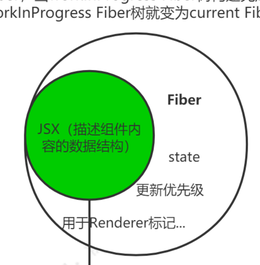
更新时，开启一次新的render阶段并构建一棵新的workInProgress Fiber 树，尝试复用current Fiber的节点，也就是diff算法

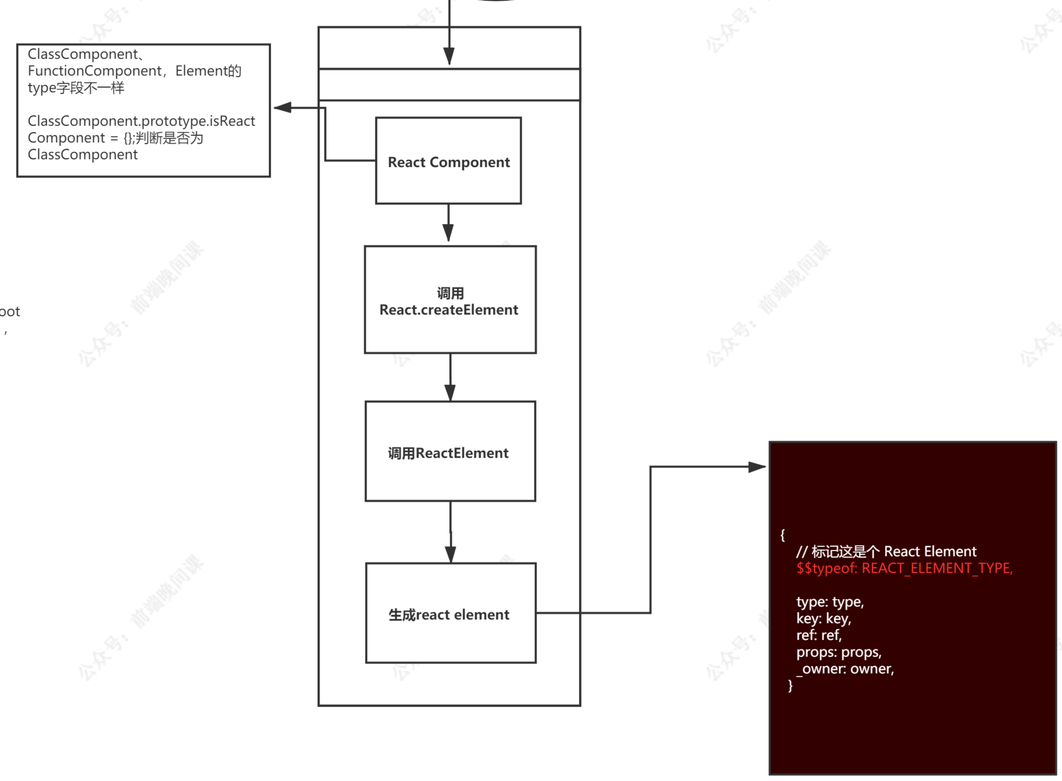


commit阶段，workInProcess Fiber树渲染成页面，改变fiberRoot current指针指向，workInProcess Fiber树变为current Fiber树，完成update

7

JSX与Fiber





8