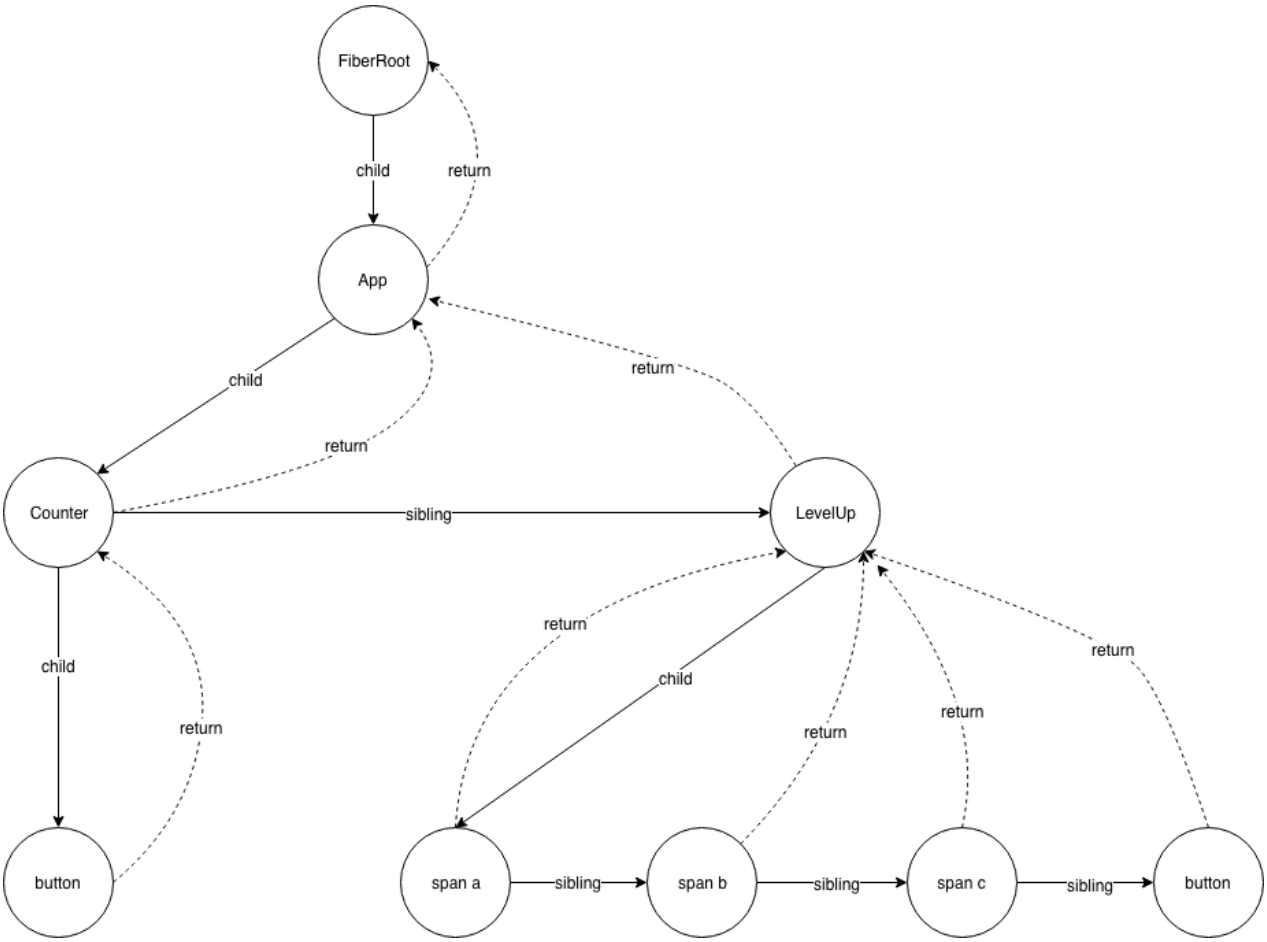
🌲2.React创建Fiber Root

2-1.ReactDOM.render

React16新增了hydrate使用服务端渲染和本地DOM进行调和。

> legacyRenderSubtreeIntoContainer具体的完成了render任务，通过_reactRootContainer标识了react节点并完成了fiberRoot创建，ReactSyncRoot来源于,初始化更新的时候采用unbatchedUpdates，需要尽快的完成渲染。



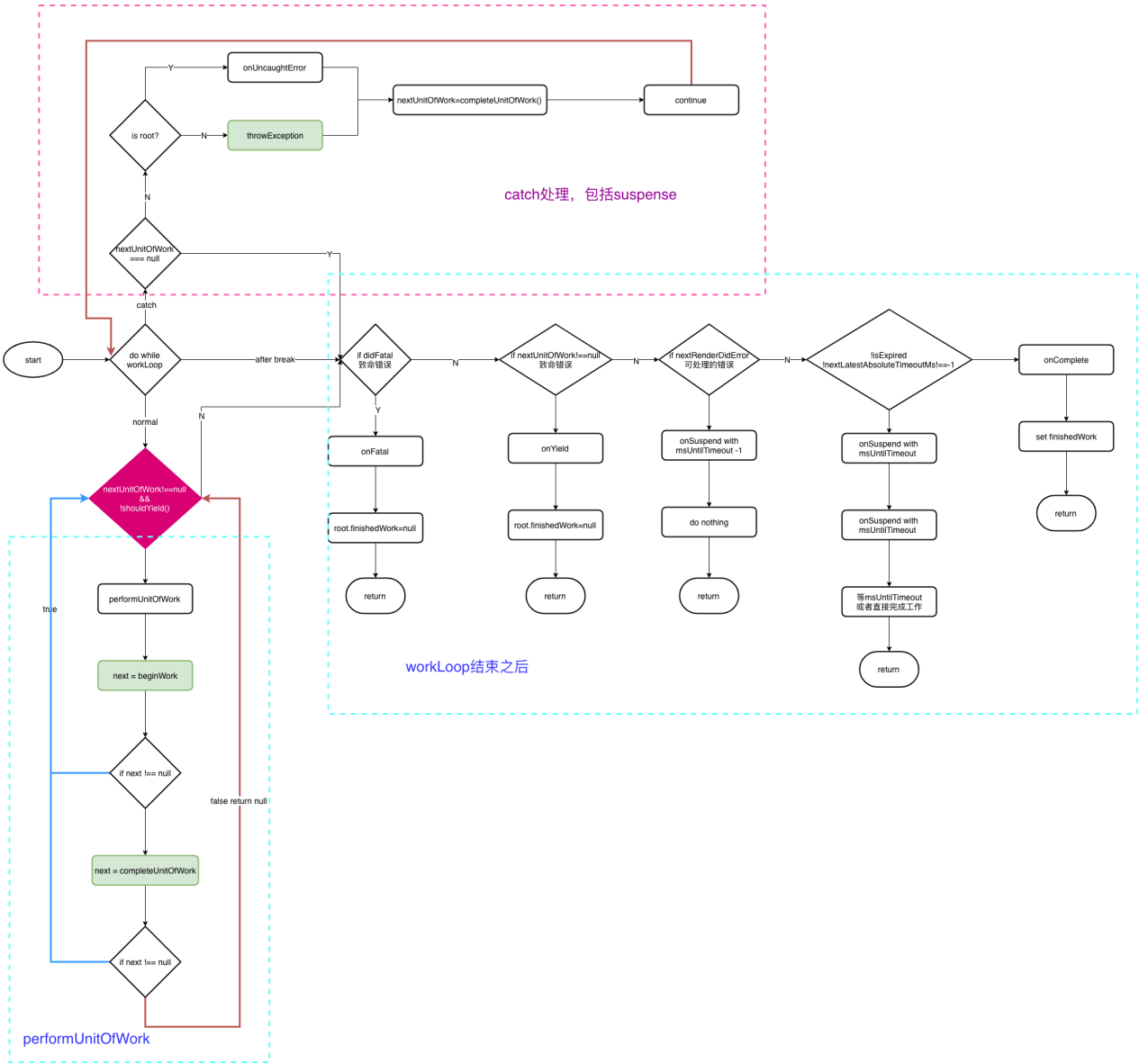
2-2.Fiber Root

> react-reconciler负责具体Fiberroot的构建,他承载了整个React更新调度全部的数据结构

```
function FiberRootNode(containerInfo, tag, hydrate) {
  // 标记不同的组件类型
  this.tag = tag;
  // 当前应用对应的Fiber对象
  this.current = null;
  // root节点
  this.containerInfo = containerInfo;
  // 只有在持久更新中会用到
```

```
  this.pendingChildren = null;
  this.pingCache = null;
  this.finishedExpirationTime = NoWork;
  // 在commit阶段只会处理这个值对应的任务
  this.finishedWork = null;
   // 在任务被挂起的时候通过setTimeout设置的返回内容，用来下一次如果有新的任务挂起时清理还没触发的timeout
  this.timeoutHandle = noTimeout;
}
```

> 创建完Fiber Root在unbatchedUpdates中执行updateContainer对容器内容进行更新，更新前会先通过expirationTime对节点结算超时时间，具体是通过在ReactFiberWorkLoop中computeExpirationForFiber进行计算，那么我们root循环调用了ReactFiberWorkLoop



> 这个超时时间实现的非常精妙，我们拿computeAsyncExpiration举例子，在computeExpirationBucket中接收的就是currentTime、5000和250最终的公式就是酱紫的：((((currentTime - 2 + 5000 / 10) / 25) | 0) + 1) * 25

```
ceiling(
      MAGIC_NUMBER_OFFSET - currentTime + expirationInMs / UNIT_SIZE,
      bucketSizeMs / UNIT_SIZE)

function ceiling(num: number, precision: number): number {
  return (((num / precision) | 0) + 1) * precision;
}
```

> 翻译一下就是：最终结果是以25为单位向上增加的，比如说我们输入10002 - 10026之间，最终得到的结果都是10525，但是到了10027的到的结果就是10550，这就是除以25取整的效果。
>
> 其实一句话 100 / 25 | 0 = 4; 4|0 = 0100 | 0000 =>0100
>
> 101 / 25 | 0 = 4; 4|0 = 0100 | 0000 =>0100

React 这么设计抹相当于抹平了25ms内计算过期时间的误差，那他为什么要这么做呢？看到LOW_PRIORITY_BATCH_SIZE这个字样，bacth，是不是就对应batchedUpdates？再细想了一下，这么做也许是为了让非常相近的两次更新得到相同的expirationTime，然后在一次更新中完成，相当于一个自动的batchedUpdates。神奇不神奇？

志佳老师@2019