思维导图

事件循环 !!!!

Node 有四个主要类型的队列，被 livuv 事件循环处理。

- 到期的 setTimeout 和 setInterval 回调队列
- I/O 事件队列
- Immediate 回调队列
- close 事件的回调队列
- process.nextTick 创建的 nextTick 回调队列
- 微任务队列

Node 事件循环分为 6 个阶段

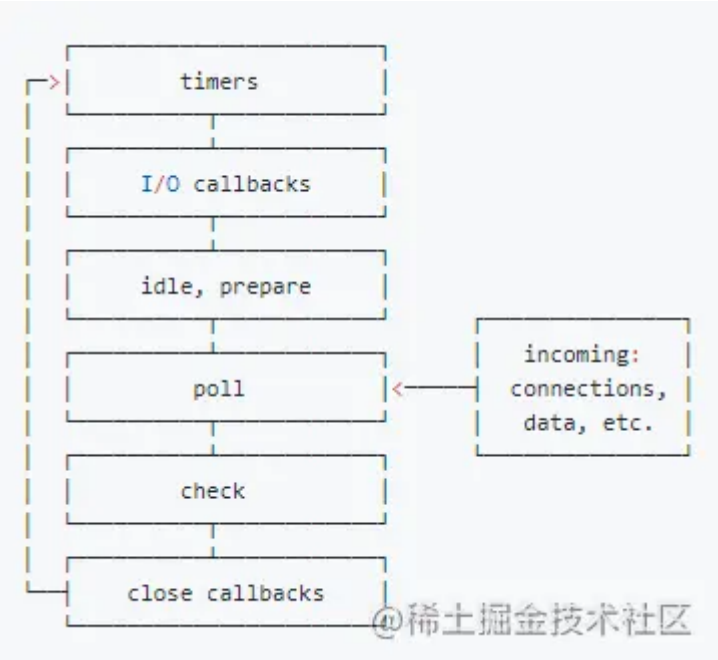
- `timer` 定时器：执行到期的 `setTimeout` 和 `setInterval` 的回调
- `I/O callback` 待定回调：处理上一轮循环中少数未执行的 I/O 回调
- `idle, prepare`：仅在 node 内部使用

- `poll` 轮询：检索新的 I/O 事件；执行与 I/O 相关的回调，其余情况 node 将在适当时候在轮询阶段阻塞，等待事件回调加入并立即执行回调，为了防止阻塞时间过长会通过一个定时器来终止阻塞。
  - 如果轮询队列不是空的，事件循环将循环访问回调队列并同步执行回调函数，直到队列为空。
  - 如果轮询队列为空时，会先检测是否有被 setImmediate 调度的函数，如果有事件循环将结束轮询阶段，开始 check 检查阶段执行被 setImmediate 调度的脚本。如果脚本未被 setImmediate 调度，则事件循环将等待回调被添加到队列中，并立即执行。
  - 一旦轮询队列为空，事件循环将检查是否有已经到期的定时器的回调，如果有就会切换到 timer 阶段以执行这些定时器的回调。

```
setTimeout(() => {
  console.log("setTimeout");
}, 0);
setImmediate(() => {
  console.log("setImmediate");
});
```

  node 在执行到 setTimeout 时如果发现 设定的时间为 0 会强制设定为 1 毫秒。

  在最开始执行同步代码时如果执行的时间比较长超过了 1 毫秒，就会先执行 setTimeout。如果小于 1 毫秒就会先跳过 timer - check 之间空的阶段先行执行 setImmediate。
- `check` 检查：执行 `setImmediate` 的回调函数
- `close callbacks` close 事件的回调函数：执行 `close` 事件的回调，例如 `socket.on('close'[, fn])` 或者 `http.servr.on('close'[,fn])`



总结：

外部输入数据 -> 轮询 poll -> 检查 check -> 关闭事件回调 clock callbacks
-> 定时器检查阶段 timers -> I/O 事件回调阶段 -> 闲置阶段 idle，prepare -> poll

Node.js 中 微任务队列会在每个阶段执行完后清空执行，且 node 中的微任务 `process.nextTick` 优先级最高。

Node 11 版本之前与 Node 11 版本之后的事件循环机制不同，Node 11 版本之前只要是要执行宏任务的话，会在当前等级的到期的宏任务执行完后才能轮到微任务，而如果是 11 版本之后的话事件循环机制与浏览器类似，每执行完一个宏任务就会清空微任务。