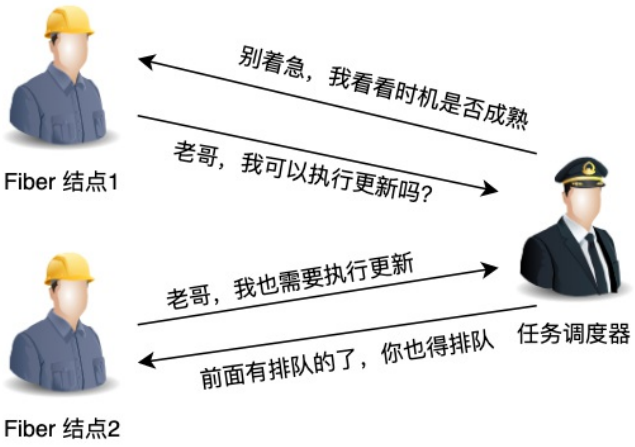
17 从设计者的角度理解 React 中的任务与任务队列

更新时间：2020-09-02 15:47:48

> 勤学如春起之苗，不见其增，日有所长。——陶潜

前言

React 中的任务是由任务调度器（scheduler）统一管理。任务调度器（scheduler）是独立于 react 和 react-dom 的模块。React 会使用调度器（scheduler）模块暴露出的一些方法安排任务执行。



任务调度器可以观测浏览器的运行情况，只有浏览器空闲的时候才能让 Fiber 结点去执行它的更新。那么，任务调度器是如何定义任务以及管理任务（如将任务加入到任务队列）的呢？

任务调度器对任务的定义

任务调度器（scheduler）中对任务做了如下的定义，见代码示例 4.2.1。

```
// 源码位置：packages/scheduler/src/Scheduler.js
var task = {
  // 任务的回调函数，主要用于和其他框架的链接，比如React fiber
  callback,
  // 任务优先级，数值越小优先级别越高，具体取值见代码示例 4.2.2
  priorityLevel,
  // 任务开始执行时间
  startTime,
  // 任务过期时间，具体取值见代码示例4-2-3
  expirationTime,
  // 下一个任务
  next: null,
  // 上一个任务
  previous: null,
};
```

代码示例 4.2.1 任务的定义

每个任务都有自己的执行优先级，任务调度器（scheduler）对任务优先级的取值范围做了如下的定义，见代码示例 4.2.2。

> 注：任务数据结构中的callback很重要，后面会有详细介绍。

```
// 源码位置：packages/scheduler/src/Scheduler.js
// 立即执行（可由饥饿任务转换），最高优先级
var ImmediatePriority = 1;
// 用户阻塞级别（如外部事件），次高优先级
var UserBlockingPriority = 2;
// 普通优先级
var NormalPriority = 3;
// 低优先级
var LowPriority = 4;
// 最低优先级，空闲时去执行
var IdlePriority = 5;
```

代码示例 4.2.2 任务优先级的取值范围

任务调度器（scheduler）对任务过期时间的取值范围做了如下的定义，见代码示例 4.2.3。

```
// 源码位置：packages/scheduler/src/Scheduler.js
// 优先级-立即执行
// 32位系统在V8中的最大整数大小 Math.pow(2, 30) - 1 = 0b111111111111111111111111111111
var maxSigned31BitInt = 1073741823;
// 立即过期/饥饿的任务
var IMMEDIATE_PRIORITY_TIMEOUT = -1;
// 用户阻塞任务的过期时间250ms
var USER_BLOCKING_PRIORITY = 250;
// 正常任务的过期时间5000ms
var NORMAL_PRIORITY_TIMEOUT = 5000;
// 低优先级任务的过期时间10000ms
var LOW_PRIORITY_TIMEOUT = 10000;
// 最低优先级任务的过期时间maxSigned31BitInt（大约12天后过期）
var IDLE_PRIORITY = maxSigned31BitInt;
```

代码示例 4.2.3

任务调度器会通过 `performance.now() + timeout` 来计算出任务的过期时间，随着时间的推移，当前时间会越来越接近这个过期时间，所以过期时间越小的代表优先级越高。如果过期时间已经比当前时间小了，说明这个任务已经过期了还没执行，需要立马去执行。

> 任务的过期时间和「更新」的过期时间是两个层级，任务的过期时间是在任务体系里面用于监控任务的时间，它和任务优先级是强相关的，其过期时间越小，表示该任务的执行优先级就越高，越应该优先被执行。
>
> `window.performance.now` 是浏览器内置的时钟，从页面加载开始计时，返回到当前的总时间，单位 `ms` 。比如我们打开某个页面10分钟后，在控制台执行该函数，得到的值是600000。

根据过期时间将任务加入到任务队列

任务调度器（`scheduler`）为任务维护了一个队列，这个队列的结构是**双向循环链表**，任务加入任务队列（taskQueue）的过程也是**将所有任务根据过期时间进行排序的过程**，其逻辑见代码示例 4.2.4。

```
// 源码位置：packages/scheduler/src/Scheduler.js
// 将新任务插入到任务队列是会根据任务的超时/过期时间排序，如果队列中已经有了与新任务相同超时/过期时间的任务，则
// 将新任务插入到它们的后面
function insertScheduledTask(newTask, expirationTime) {
  if (firstTask === null) {
    // 如果firstTask为null，那么newTask成为任务队列中的第一个任务
    firstTask = newTask.next = newTask.previous = newTask;
  } else {
    var next = null;
    var task = firstTask;
    // 遍历任务链表
    do {
      if (expirationTime < task.expirationTime) {
        // 如果新任务的过期时间小于链表中某个任务的过期时间，说明已经找到了任务链表中要插入的位置，则跳出循环
        next = task;
        break;
      }
      task = task.next;
    } while (task !== firstTask);

    if (next === null) {
      // 任务链表遍历结束，next指针为null，说明任务链表中的任务过期时间均比新任务的过期时间大，则将新任务插入到链表尾部
      next = firstTask;
    } else if (next === firstTask) {
      // 如果next指向firstTask，说明新任务在整个链表中拥有最小的过期时间
      firstTask = newTask;
    }

    // 操作任务链表，将新任务插入到合适的位置
    var previous = next.previous;
    previous.next = next.previous = newTask;
    newTask.next = next;
    newTask.previous = previous;
  }
}
```

代码示例 4.2.4 任务加入到任务队列

将任务插入到任务队列其实就是一个处理双向链表的过程，在这个过程中使用`task = task.next`走向下一个结点，当找到正确的结点位置时使用`next`和`previous`指针将新结点和链表中已有的结点串联起来。任务加入到任务队列后，任务队列（taskQueue）会形成一个双向循环链表，其内部结构如图 4.2.1。

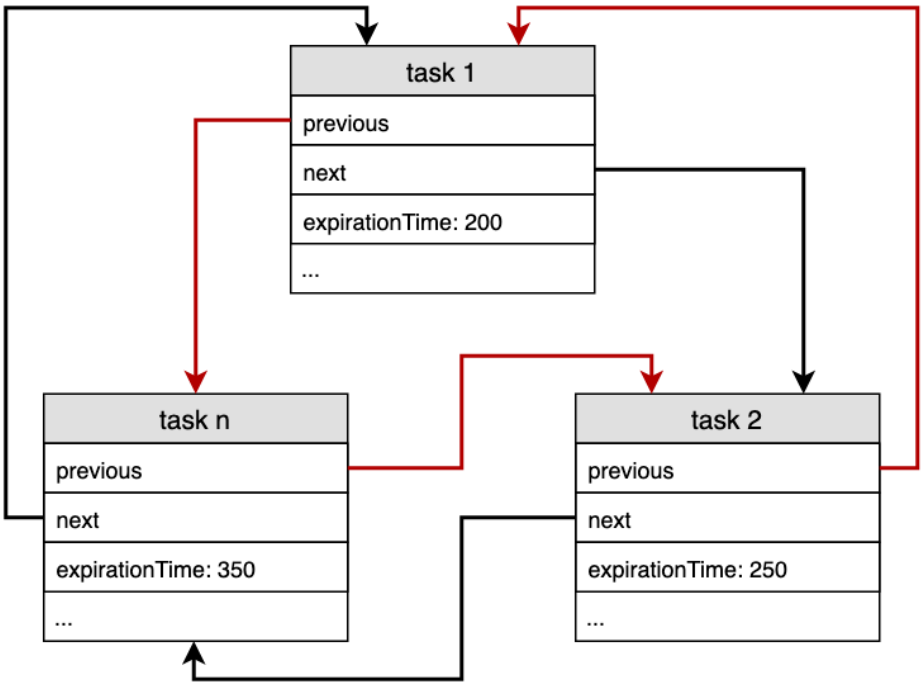


图 4.2.1 任务队列（taskQueue）数据结构

任务调度器中维护的任务队列在内存中的结构是一个双向循环链表，每个 task 对象可以通过`previous`和`next`找到其上一个任务和下一个任务。

小结

「任务」是一个抽象的概念，React 给「任务」定义了数据结构，使其具有过期时间、下一个任务和上一个任务等属性，多个任务对象链接成一个双向循环链表。

本节介绍任务偏向于整体层面，如任务的定义、任务队列的结构及将任务加入到任务队列的逻辑。有关「任务的具体内容」并没有详细介绍。那么，任务的具体内容到底是什么呢？在揭晓答案之前我们在下一节中先来了解一下任务调度器是如何和浏览器进行通信的。

}

16 如何理解 React 中的更新与任务

18 React 中的任务调度器（scheduler）