16 如何理解 React 中的更新与任务

更新时间：2020-08-31 09:47:25

> 如果不想在世界上虚度一生，那就要学习一辈子。——高尔基

前言

我们仔细想一想，【更新】和【任务】是两个抽象的概念，从字面上理解这两个概念应该是静态的。在第二章中介绍了 React 对「更新」对象的定义，在本章也会介绍 React 对「任务」对象的定义。那么，在 React 中「更新」和「任务」有什么关系吗？我们该如何理解它们呢？

事实上，「更新」和「任务」和是有关系的，但是两者也属于不同的理论体系。「更新」作用于 React Fiber 结点，它是 Fiber 架构的一部分。「任务」作用于任务调度器，是任务体系里面的一部分。**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。**

它们两者之间什么时候才会产生碰撞呢？**React 将「更新」内容映射到屏幕的过程就是执行更新任务的过程。**

> 上面说到的「更新」和「任务」均为静态的，React 对它们都定义了具体的数据结构，也就是说它们都有实体对象，而「执行任务」是一个动态的过程。「执行任务」不也是抽象的吗？那么，对于 React 来说怎么才算是执行一个任务呢？不要着急，后面会有介绍。

回顾 React 对更新的定义

```
var update = {
  // 过期时间与任务优先级相关联
  expirationTime: expirationTime,
  suspenseConfig: suspenseConfig,
  // tag用于标识更新的类型如UpdateState，ReplaceState，ForceUpdate等
  tag: UpdateState,
  // 更新内容
  payload: null,
  // 更新完成后的回调
  callback: null,
  // 下一个更新（任务）
  next: null,
  // 下一个副作用
  nextEffect: null
};
```

代码示例 4.1.1 React 对更新的定义

现在，我们重点探讨一下更新对象的 expirationTime （过期时间）。在 React 中，为防止某个 update 因为优先级的原因一直被打断而未能执行，React 会设置一个 expirationTime，当时间到了 expirationTime 的时候，如果某个 update 还未执行的话，React 将会强制执行该 update，这就是 expirationTime 的作用。那么，一个更新对象的过期时间可以取哪些值，它又是怎么得来的呢？

React 计算更新对象过期时间的流程

首先，我们来看一下更新对象的过期时间可以取哪些值，见代码示例 4.1.2。

```
// 定义了过期时间的最大值Math.pow(2, 30) - 1
// 0b111111111111111111111111111111
var MAX_SIGNED_31_BIT_INT = 1073741823;
// NoWork表示不需要更新
var NoWork = 0;
// Never表示更新的过期时间很小，可以无限被延期
var Never = 1;
// Sync表示拥有最大过期时间的更新需要立即执行，也就是同步执行
var Sync = MAX_SIGNED_31_BIT_INT;
// Batched表示批量更新的过期时间
var Batched = Sync - 1;

var UNIT_SIZE = 10;
var MAGIC_NUMBER_OFFSET = Batched - 1;
```

代码示例 4.1.2 更新对象的过期时间取值

当组件内部触发了 setState(...) 操作后，组件更新器会调用 enqueueSetState 函数，在该函数内部为当前 Fiber 结点的更新创建过期时间，见代码实例 4.1.3。

```
var classComponentUpdater = {
 enqueueSetState: function (inst, payload, callback) {
   var fiber = get(inst);
   var currentTime = requestCurrentTime();
   // 为更新对象计算过期时间
   var expirationTime = computeExpirationForFiber(currentTime, fiber, suspenseConfig);
   // 根据过期时间创建更新对象
   var update = createUpdate(expirationTime, suspenseConfig);
   update.payload = payload;
 },
 // ...
}
```

代码示例 4.1.3 为更新对象计算过期时间

`computeExpirationForFiber` 函数的作用是为 Fiber 结点计算过期时间。那么，它是根据什么来计算的呢？

通过任务优先级计算更新的过期时间

React 在计算当前 Fiber 结点更新对象的过期时间时，会向其任务调度器查询该更新对象应该什么样的优先级。任务调度器会根据当前计算机资源（如 CPU）的使用情况返回合适的优先级。那么，任务的优先级都有哪些呢？见代码示例 4.1.4。

```
// 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 4.1.4 React 任务优先级取值

前面提到，`computeExpirationForFiber` 函数的作用是为 Fiber 结点计算过期时间，在 `computeExpirationForFiber` 函数里面调用 `getCurrentPriorityLevel` 函数可获取当前更新任务的优先级，见代码示例 4.1.5。

```
// 源码位置：packages/react-reconciler/src/ReactFiberWorkLoop.js
function computeExpirationForFiber(currentTime, fiber, suspenseConfig) {
  var mode = fiber.mode;
  // 如果处于非批量更新模式直接按照同步任务执行
  if ((mode & BatchedMode) === NoMode) {
    return Sync;
  }
  // ...
  var priorityLevel = getCurrentPriorityLevel();
  // 根据调度器的任务优先级计算过期时间
  switch (priorityLevel) {
    // ImmediatePriority对应的数值为99
    case ImmediatePriority:
      expirationTime = Sync;
      break;
    // UserBlockingPriority对应的数值为98
    case UserBlockingPriority:
      // 计算交互事件的过期时间
      expirationTime = computeInteractiveExpiration(currentTime);
      break;
    // NormalPriority对应的数值为97，为初始值
    case NormalPriority:
    // LowPriority对应的数值为96
    case LowPriority:
      // 计算异步更新的过期时间
      expirationTime = computeAsyncExpiration(currentTime);
      break;
    // IdlePriority对应的数值为95
    case IdlePriority:
      expirationTime = Never;
      break;
    default:
      throw ReactError(Error('Expected a valid priority level'));
  }
  // ...
}
```

代码示例 4.1.5 计算更新的过期时间

`getCurrentPriorityLevel` 函数是任务调度器暴露出来的方法，该函数可获取调度器中当前任务的优先级，取值有 `ImmediatePriority`、`UserBlockingPriority`、`NormalPriority`、`LowPriority` 和 `IdlePriority`，其中 `NormalPriority` 为默认值。然后在 `computeExpirationForFiber` 函数内部根据当前任务优先级为 Fiber 结点匹配对应的过期时间。

> 注：`react-dom` 和 `scheduler` 是两个独立的模块，任务的优先级主要由 `scheduler` 模块定义，但是在 `react-dom` 模块也用变量维护了一个优先级备份，这个备份与 `scheduler` 模块的关系就是 `100 - scheduler.priority`。在本章第四节将会介绍 `getCurrentPriorityLevel` 函数是如何返回任务优先级的。

我们想一想，为什么要将调度器中当前任务的优先级返回给未执行更新的 Fiber 结点呢？

任务调度器拥有任务调度权利，它可以监听到浏览器当前的任务执行状态，比如是否有交互事件，如果有交互事件则将当前任务优先级（`currentPriorityLevel`）设置为 `UserBlockingPriority`。也就是说接下来 Fiber 结点是要根据用户交互行为阻塞情况选择适当时机执行更新的。如果任务调度器返回的是 `ImmediatePriority`，一般情况是由于 Fiber 结点的更新任务一直得不到执行，任务调度器将其改变为饥饿任务并为其提高优先执行权。

下面我们通过一个例子来分析一下 `computeInteractiveExpiration` 函数是怎么计算交互事件中 Fiber 结点更新的过期时间，见代码示例 4.1.6。

```
// 下面是React定义的一些常量，单位均为ms
var HIGH_PRIORITY_EXPIRATION = 500;
var HIGH_PRIORITY_BATCH_SIZE = 100;
var LOW_PRIORITY_EXPIRATION = 5000;
var LOW_PRIORITY_BATCH_SIZE = 250;
var MAX_SIGNED_31_BIT_INT = 1073741823;
var Sync = MAX_SIGNED_31_BIT_INT;
var Batched = Sync - 1;
var MAGIC_NUMBER_OFFSET = Batched - 1;
var UNIT_SIZE = 10;

function ceiling(num, precision) {
 // | 0 为取整操作
  return ((num / precision | 0) + 1) * precision;
}
// 计算过期时间bucket
function computeExpirationBucket(currentTime, expirationInMs, bucketSizeMs) {
  return MAGIC_NUMBER_OFFSET - ceiling(MAGIC_NUMBER_OFFSET - currentTime + expirationInMs / UNIT_SIZE, bucketSizeMs / UNIT_SIZE);
}
// 计算交互事件过期时间的入口函数
function computeInteractiveExpiration(currentTime) {
  return computeExpirationBucket(currentTime, HIGH_PRIORITY_EXPIRATION, HIGH_PRIORITY_BATCH_SIZE);
}
```

代码示例 4.1.6 计算交互事件更新的过期时间

我们把代码示例 4.1.6 中的常量用数值转换一下，见代码示例 4.1.7。

```
// currentTime 一般是通过window.performance.now()获取，然后转化成React中的时间
function computeInteractiveExpiration(currentTime) {
  return computeExpirationBucket(currentTime, 500, 100);
}

function computeExpirationBucket(currentTime, 500, 100) {
  // return 1073741821 - ceiling(1073741821 - currentTime + 500 / 10, 100 / 10);
  // return 1073741821 - ceiling(1073741821 - currentTime + 50, 10);
  return 1073741821 - ((1073741871 - currentTime) / 10 | 0) * 10 + 10
  // 如果currentTime的值为 1073738202 过期时间为 1073738171
  // 如果currentTime的值为 1073738211 过期时间依然为 1073738171
  // 如果currentTime的值为 1073738212 过期时间依然为 1073738181
}
```

代码示例 4.1.7 计算交互事件更新的过期时间代码简化

通过对 currentTime 分别取不同对值进行计算后发现，具有 UserBlockingPriority 级别的多个更新，如果它们的时间间隔小于 10ms，那么它们拥有相同的过期时间。同样的方式可以推到出具有 LowPriority 级别的多个更新（一般为异步更新），如果它们的时间间隔小于 25ms，那么它们也拥有相同的过期时间。**React 的过期时间机制保证了短时间内同一个 Fiber 结点的多个更新拥有相同的过期时间，最终会合并在一起执行。**

> 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。

小结

本节从两个层面讲述了「更新」与「任务」之间的关系，「更新」是 Fiber 架构中的一部分，而「任务」是任务调度器体系中的一部分。React 将 Fiber 结点中的「更新」映射到屏幕的过程就是执行更新任务的过程，但是需要先获取任务调度器为其分配的任务执行优先级。任务执行优先级是「更新」过期时间是密切相关的，Fiber 结点的更新被分配的任务优先级越高，其过期时间也就越大。

本节中尚未展开介绍的就是 React 任务调度器体系的任务和任务队列，下一节将会进行详细介绍。

}

15 理解 React Fiber 架构中中的 effectTag 与位运算

17 从设计者的角度理解 React 中的任务与任务队列