19-1 React Lanes 模型背景

二进制基础

- 二进制基础

位操作

位操作是程序设计中对位数组或二进制数的一元和二元操作。在许多古老的微处理器上，位运算比加减运算略快，通常位运算比乘除法运算要快很多。在现代架构中，位运算的运算速度通常与加法运算相同（仍然快于乘法运算），但是通常功耗较小，因为资源使用减少。

位运算符

~ 按位非

& 按位与

| 按位或

^ 按位异或

>> 右移

<< 左移

位掩码（掩码）

维基百科：https://en.wikipedia.org/wiki/Mask_(computing)

In computer science, a **mask** or **bitmask** is data that is used for bitwise operations, particularly in a bit field. Using a mask, multiple bits in a byte, nibble, word, etc. can be set either on or off, or inverted from on to off (or vice versa) in a single bitwise operation. An additional use of masking involves predication in vector processing, where the bitmask is used to select which element operations in the vector are to be executed (mask bit is enabled) and which are not (mask bit is clear).

bitmask

位指的是二进制数据当中的二进制位。

掩码，mask，在计算机学科及数字逻辑中指的是**一串二进制数字**，通过与目标数字的**按位**操作，达到屏蔽指定位而实现需求。

React 中的二进制的应用

Lanes、fiber 的 Flags、HookFlags、ReactFiberWorkLoop.js 中记录上下文环境的 ExecutionContext 都是用的二进制。

值组合：数组、Set|Map、字符串、**二进制(后面运算速度快、占用资源少、运算方便)**。

如：

Fiber Flags

react/packages/react-reconciler/src/ReactFiberFlags.js

JavaScript

```
export type Flags = number;

// Don't change these values. They're used by React Dev Tools.
export const NoFlags = /*                      */ 0b0000000000000000000000000
export const PerformedWork = /*                */ 0b0000000000000000000000001
export const Placement = /*                    */ 0b0000000000000000000000010
export const DidCapture = /*                   */ 0b0000000000000000000010000
export const Hydrating = /*                    */ 0b0000000000000000000100000

// You can change the rest (and add more).
export const Update = /*                       */ 0b0000000000000000000000100
/* Skipped value:                                 0b0000000000000000000001000

export const ChildDeletion = /*                */ 0b0000000000000000000001000
export const ContentReset = /*                 */ 0b0000000000000000000010000
export const Callback = /*                     */ 0b0000000000000000000100000
/* Used by DidCapture:                            0b0000000000000000000010000

export const ForceClientRender = /*            */ 0b0000000000000000000100000
export const Ref = /*                          */ 0b0000000000000000001000000
export const Snapshot = /*                     */ 0b0000000000000000010000000
export const Passive = /*                      */ 0b0000000000000000100000000
```

HookFlags

react/packages/react-reconciler/src/ReactHookEffectTags.js

JavaScript

```
export type HookFlags = number;
```

```
export const NoFlags = /*   */ 0b0000;

// Represents whether effect should fire.
export const HasEffect = /* */ 0b0001; // 1

// Represents the phase in which the effect (not the clean-up) fires.
export const Insertion = /* */ 0b0010; // 2
export const Layout = /*    */ 0b0100; // 4
export const Passive = /*   */ 0b1000; // 8
```

Lanes 模型背景

一个位掩码可以标记一个任务，这个位掩码我们称为一个 Lane。一个位掩码可以标记一个批量任务，这个位掩码我们称为 Lanes。

React17 开始正式采用 Lanes 模型，以前用的是 expirationTime 模型。区别在于前者是二进制数字，而后者是十进制数字。二者最主要的区别在于二进制非常适合处理批量 update。比如：

JavaScript
```
const isTaskIncludedInBatch = (task & batchOfTasks) !== 0;
```

这里可以对比 fiber 的 Flags。

lanes 与 lane 值类型

多个 lane 组合，运算之后，就会得到 lanes。

react/packages/react-reconciler/src/ReactFiberLane.js

JavaScript
```
export type Lanes = number;
export type Lane = number;
export type LaneMap<T> = Array<T>;
```

lanes 与 lane 值

react/packages/react-reconciler/src/ReactFiberLane.js

JavaScript

```
// * lane 值越小，优先级越高
export const TotalLanes = 31;

export const NoLanes: Lanes = /*                        */ 0b0000000000
export const NoLane: Lane = /*                          */ 0b0000000000

export const SyncHydrationLane: Lane = /*               */ 0b0000000000
export const SyncLane: Lane = /*                        */ 0b0000000000
export const SyncLaneIndex: number = 1;

export const InputContinuousHydrationLane: Lane = /*    */ 0b0000000000
export const InputContinuousLane: Lane = /*             */ 0b0000000000

export const DefaultHydrationLane: Lane = /*            */ 0b0000000000
export const DefaultLane: Lane = /*                     */ 0b0000000000

export const SyncUpdateLanes: Lane = enableUnifiedSyncLane
  ? SyncLane | InputContinuousLane | DefaultLane
  : SyncLane;

const TransitionHydrationLane: Lane = /*                */ 0b0000000000
const TransitionLanes: Lanes = /*                       */ 0b0000000000
const TransitionLane1: Lane = /*                        */ 0b0000000000
const TransitionLane2: Lane = /*                        */ 0b0000000000
const TransitionLane3: Lane = /*                        */ 0b0000000000
const TransitionLane4: Lane = /*                        */ 0b0000000000
const TransitionLane5: Lane = /*                        */ 0b0000000000
const TransitionLane6: Lane = /*                        */ 0b0000000000
const TransitionLane7: Lane = /*                        */ 0b0000000000
const TransitionLane8: Lane = /*                        */ 0b0000000000
const TransitionLane9: Lane = /*                        */ 0b0000000000
const TransitionLane10: Lane = /*                       */ 0b0000000000
const TransitionLane11: Lane = /*                       */ 0b0000000000
const TransitionLane12: Lane = /*                       */ 0b0000000000
const TransitionLane13: Lane = /*                       */ 0b0000000000
const TransitionLane14: Lane = /*                       */ 0b0000000000
const TransitionLane15: Lane = /*                       */ 0b0000000000
```

```
const RetryLanes: Lanes = /*                            */ 0b000001111
const RetryLane1: Lane = /*                             */ 0b000000001
const RetryLane2: Lane = /*                             */ 0b000000010
const RetryLane3: Lane = /*                             */ 0b000000100
const RetryLane4: Lane = /*                             */ 0b000001000

export const SomeRetryLane: Lane = RetryLane1;

export const SelectiveHydrationLane: Lane = /*          */ 0b000010000

const NonIdleLanes: Lanes = /*                          */ 0b000011111

export const IdleHydrationLane: Lane = /*               */ 0b000100000
export const IdleLane: Lane = /*                        */ 0b001000000

export const OffscreenLane: Lane = /*                   */ 0b010000000
export const DeferredLane: Lane = /*                    */ 0b100000000

// Any lane that might schedule an update. This is used to detect infi
// update loops, so it doesn't include hydration lanes or retries.
export const UpdateLanes: Lanes =
  SyncLane | InputContinuousLane | DefaultLane | TransitionLanes;
```

lanes 与 lane 基本运算函数

react/packages/react-reconciler/src/ReactFiberLane.js

JavaScript

```
// 是否是transitionLane
export function isTransitionLane(lane: Lane): boolean {
  return (lane & TransitionLanes) !== NoLanes;
}

// 获取优先级最高的lane
// 因为在 lane 的值中，值越小，代表的优先级越高。即
// 获取最低位的1，如4194240&-4194240就是64
// 负数原码转换为补码的方法：符号位保持1不变，数值位按位求反，末位加1
export function getHighestPriorityLane(lanes: Lanes): Lane {
  return lanes & -lanes;
```

```
}
// 是否包含某个lane或者某些lane(lanes)
export function includesSomeLane(a: Lanes | Lane, b: Lanes | Lane): bo
  return (a & b) !== NoLanes;
}

// set是否包含subset，和 includesSomeLane 不同，includesSomeLane检查的是a和
// 而这里的 isSubsetOfLanes 检查的是 subset 是否是 set 的子集
export function isSubsetOfLanes(set: Lanes, subset: Lanes | Lane): boo
  return (set & subset) === subset;
}

// 合并两个lane或者lanes
export function mergeLanes(a: Lanes | Lane, b: Lanes | Lane): Lanes {
  return a | b;
}
// 移除某个lane或者lanes，比如执行完节点的 Update 操作之后，则需要移动 fiber.f
export function removeLanes(set: Lanes, subset: Lanes | Lane): Lanes {
  return set & ~subset;
}
```