



19-2 React Lanes 模型的应用

Lane

位掩码，用于标记一个任务的优先级。每个 lane 上只有一个比特位为 1。

Update 的 lane

每个 update 会对应一个 lane。

所以在 Update 上有个属性：

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

```
export type Update<State> = {  
  lane: Lane,  
  ...  
};
```

JavaScript

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

```
export type Update<S, A> = {  
  lane: Lane,  
  revertLane: Lane,
```

JavaScript

```

    action: A,
    hasEagerState: boolean,
    eagerState: S | null,
    next: Update<S, A>,
};

```

事件优先级

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

JavaScript

```
export opaque type EventPriority = Lane;

export const DiscreteEventPriority: EventPriority = SyncLane;
export const ContinuousEventPriority: EventPriority = InputContinuousLane;
export const DefaultEventPriority: EventPriority = DefaultLane; // 页面
export const IdleEventPriority: EventPriority = IdleLane;

let currentUpdatePriority: EventPriority = NoLane;
```

调度更新 scheduleUpdateOnFiber

```

DebugReact > src > react > packages > react-reconciler > src > JS ReactFiberHooks.js > ...
3254 update.eagerState = eagerState;
3255 if (!isEagerState, currentState) {
3256   // Fast path. We can bail out without scheduling React to re-render.
3257   // It's still possible that we'll need to rebase this update later,
3258   // if the component re-renders for a different reason and by that
3259   // time the reducer has changed.
3260   // TODO: Do we still need to entangle transitions in this case?
3261   // * 如果state没变，组件不做更新，此处useReducer对比下，useReducer还是会让函数组件更新
3262   enqueueCurrentHookUpdateAndEagerlyBailout(fiber, queue, update);
3263   return;
3264 }
3265 } catch (error) {
3266   // Suppress the error. It will throw again in the render phase.
3267 } finally {
3268   if (___DEV___) {
3269     ReactCurrentDispatcher.current = prevDispatcher;
3270   }
3271 }
3272 }
3273 }
3274 // ! 2. 把update都存入currentQueues数组中 You, 13分钟前 · Uncommitted changes
3275 const root = enqueueCurrentHookUpdate(fiber, queue, update, lane);
3276 if (root !== null) {
3277   // 3. 调度更新
3278   scheduleUpdateOnFiber(root, fiber, lane);
3279   entangleTransitionUpdate(root, queue, lane);
3280 }
3281 }

```

```

DebugReact > src > react > packages > react-reconciler > src > > ReactFiberWorkLoop.js > scheduleUpdateOnFiber
740 export function scheduleUpdateOnFiber{
741   root: FiberRoot,
742   fiber: Fiber,
743   lane: Lane,
744 } {
745
746   console.log('%c [ scheduleUpdateOnFiber ]-746', 'font-size:13px; background:#d711e1; color:#000;', )
747   // Check if the work loop is currently suspended and waiting for data to
748   // finish loading.
749   if (
750     // Suspended render phase
751     (root === workInProgressRoot &&
752      workInProgressSuspendedReason === SuspendedOnData) ||
753     // Suspended commit phase
754     root.cancelPendingCommit !== null
755   ) {
756     // Mark that the root has a pending update.
757     markRootUpdated(root, lane);
758   }
759
760   if (
761     (executionContext & RenderContext) !== NoLanes &&
762     root === workInProgressRoot
763   ) {
764     //
765     //
766     //
767     //
768     //
769     //
770     //
771     //
772     //
773     //
774     //
775     //
776     //
777     //
778     //
779     //
780     //
781     //
782     //
783     //
784     //
785     //
786     //
787     //
788     //
789     //
790     //
791     //
792     //
793     //
794     //
795     //
796     //
797     //
798     //
799     //
800     //
801     //
802     //
803     //
804     //
805     //
806     //
807     //
808     //
809     //
810     //
811     //
812     //
813     //
814     //
815     //
816     //
817     //
818     //
819     //
820     //
821     //
822     //
823     //
824     //
825     //
826     //
827     //
828     //
829     //
830     //
831     //
832     //
833     //
834     //
835     //
836     //
837     //
838     //
839     //
840     //
841     //
842     //
843     //
844     //
845     //
846     //
847     //
848     //
849     //
850     //
851     //
852     //
853     //
854     //
855     //
856     //
857     //
858     //
859     //
860     //
861     //
862     //
863     //
864     //
865     //
866     //
867     //
868     //
869     //
870     //
871     //
872     //
873     //
874     //
875     //
876     //
877     //
878     //
879     //
880     //
881     //
882     //
883     //
884     //
885     //
886     //
887     //
888     //
889     //
890     //
891     //
892     //
893     //
894     //
895     //
896     //
897     //
898     //
899     //
900     //
901     //
902     //
903     //
904     //
905     //
906     //
907     //
908     //
909     //
910     //
911     //
912     //
913     //
914     //
915     //
916     //
917     //
918     //
919     //
920     //
921     //
922     //
923     //
924     //
925     //
926     //
927     //
928     //
929     //
930     //
931     //
932     //
933     //
934     //
935     //
936     //
937     //
938     //
939     //
940     //
941     //
942     //
943     //
944     //
945     //
946     //
947     //
948     //
949     //
950     //
951     //
952     //
953     //
954     //
955     //
956     //
957     //
958     //
959     //
960     //
961     //
962     //
963     //
964     //
965     //
966     //
967     //
968     //
969     //
970     //
971     //
972     //
973     //
974     //
975     //
976     //
977     //
978     //
979     //
980     //
981     //
982     //
983     //
984     //
985     //
986     //
987     //
988     //
989     //
990     //
991     //
992     //
993     //
994     //
995     //
996     //
997     //
998     //
999     //
1000    //
1001    //
1002    //
1003    //
1004    //
1005    //
1006    //
1007    //
1008    //
1009    //
1010    //
1011    //
1012    //
1013    //
1014    //
1015    //
1016    //
1017    //
1018    //
1019    //
1020    //
1021    //
1022    //
1023    //
1024    //
1025    //
1026    //
1027    //
1028    //
1029    //
1030    //
1031    //
1032    //
1033    //
1034    //
1035    //
1036    //
1037    //
1038    //
1039    //
1040    //
1041    //
1042    //
1043    //
1044    //
1045    //
1046    //
1047    //
1048    //
1049    //
1050    //
1051    //
1052    //
1053    //
1054    //
1055    //
1056    //
1057    //
1058    //
1059    //
1060    //
1061    //
1062    //
1063    //
1064    //
1065    //
1066    //
1067    //
1068    //
1069    //
1070    //
1071    //
1072    //
1073    //
1074    //
1075    //
1076    //
1077    //
1078    //
1079    //
1080    //
1081    //
1082    //
1083    //
1084    //
1085    //
1086    //
1087    //
1088    //
1089    //
1090    //
1091    //
1092    //
1093    //
1094    //
1095    //
1096    //
1097    //
1098    //
1099    //
1100    //
1101    //
1102    //
1103    //
1104    //
1105    //
1106    //
1107    //
1108    //
1109    //
1110    //
1111    //
1112    //
1113    //
1114    //
1115    //
1116    //
1117    //
1118    //
1119    //
1120    //
1121    //
1122    //
1123    //
1124    //
1125    //
1126    //
1127    //
1128    //
1129    //
1130    //
1131    //
1132    //
1133    //
1134    //
1135    //
1136    //
1137    //
1138    //
1139    //
1140    //
1141    //
1142    //
1143    //
1144    //
1145    //
1146    //
1147    //
1148    //
1149    //
1150    //
1151    //
1152    //
1153    //
1154    //
1155    //
1156    //
1157    //
1158    //
1159    //
1160    //
1161    //
1162    //
1163    //
1164    //
1165    //
1166    //
1167    //
1168    //
1169    //
1170    //
1171    //
1172    //
1173    //
1174    //
1175    //
1176    //
1177    //
1178    //
1179    //
1180    //
1181    //
1182    //
1183    //
1184    //
1185    //
1186    //
1187    //
1188    //
1189    //
1190    //
1191    //
1192    //
1193    //
1194    //
1195    //
1196    //
1197    //
1198    //
1199    //
1200    //
1201    //
1202    //
1203    //
1204    //
1205    //
1206    //
1207    //
1208    //
1209    //
1210    //
1211    //
1212    //
1213    //
1214    //
1215    //
1216    //
1217    //
1218    //
1219    //
1220    //
1221    //
1222    //
1223    //
1224    //
1225    //
1226    //
1227    //
1228    //
1229    //
1230    //
1231    //
1232    //
1233    //
1234    //
1235    //
1236    //
1237    //
1238    //
1239    //
1240    //
1241    //
1242    //
1243    //
1244    //
1245    //
1246    //
1247    //
1248    //
1249    //
1250    //
1251    //
1252    //
1253    //
1254    //
1255    //
1256    //
1257    //
1258    //
1259    //
1260    //
1261    //
1262    //
1263    //
1264    //
1265    //
1266    //
1267    //
1268    //
1269    //
1270    //
1271    //
1272    //
1273    //
1274    //
1275    //
1276    //
1277    //
1278    //
1279    //
1280    //
1281    //
1282    //
1283    //
1284    //
1285    //
1286    //
1287    //
1288    //
1289    //
1290    //
1291    //
1292    //
1293    //
1294    //
1295    //
1296    //
1297    //
1298    //
1299    //
1300    //
1301    //
1302    //
1303    //
1304    //
1305    //
1306    //
1307    //
1308    //
1309    //
1310    //
1311    //
1312    //
1313    //
1314    //
1315    //
1316    //
1317    //
1318    //
1319    //
1320    //
1321    //
1322    //
1323    //
1324    //
1325    //
1326    //
1327    //
1328    //
1329    //
1330    //
1331    //
1332    //
1333    //
1334    //
1
```

Lanes

几个 Lane 位运算出来的位掩码，称为 Lanes。每个 lanes 上至少有一个比特位为 1。

UpdateQueue 的 lanes

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

JavaScript

```
export type SharedQueue<State> = {
  pending: Update<State> | null, // 单向循环链表
  lanes: Lanes,
  // 如果类组件是Activity(以前叫OffScreen)的后代组件，需要延迟执行的其setState
  // Activity目前还是unstable，了解即可~
  hiddenCallbacks: Array<() => mixed> | null,
};

export type UpdateQueue<State> = {
  baseState: State,
  // 单链表 firstBaseUpdate->...->lastBaseUpdate
  firstBaseUpdate: Update<State> | null,
  lastBaseUpdate: Update<State> | null,
  shared: SharedQueue<State>,
  callbacks: Array<() => mixed> | null,
};
```

Hooks UpdateQueue 的 lanes

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

JavaScript

```
export type UpdateQueue<S, A> = {
  pending: Update<S, A> | null,
  lanes: Lanes,
  dispatch: (A => mixed) | null,
  lastRenderedReducer: ((S, A) => S) | null,
  lastRenderedState: S | null,
};
```

Fiber 的 lanes

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

JavaScript

```
export type Fiber = {
  ...
```

```

    lanes: Lanes,
    childLanes: Lanes, // 后代fiber的lanes
    // 依赖，比如context
    dependencies: Dependencies | null,
    ...
};

export type ContextDependency<T> = {
    context: ReactContext<T>,
    next: ContextDependency<mixed> | null,
    memoizedValue: T,
    ...
};

export type Dependencies = {
    lanes: Lanes,
    firstContext: ContextDependency<mixed> | null,
    ...
};

```

因为一个 fiber 上可以有多个 update，而每个 update 可以有不同的优先级。

FiberRoot 的 lanes

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

```

// updated lanes
pendingLanes: Lanes,
suspendedLanes: Lanes,
pingedLanes: Lanes,
expiredLanes: Lanes,
errorRecoveryDisabledLanes: Lanes,
shellSuspendCounter: number,

finishedLanes: Lanes,

// 记录当前更新与其他更新之间的关联性，即它们之间存在依赖或相关性。
// 当一个更新被触发时，React 会根据其依赖关系计算出 entangledLanes，这些 ent

```

JavaScript

```
entangledLanes: Lanes,  
entanglements: LaneMap<Lanes>,
```