# 17-4 React 中的事件派发

DebugReact > src > react > packages > react-dom-bindings > src > events > JS DOMPluginEventSystem.js > ...

```
437  function addTrappedEventListener(
438    targetContainer: EventTarget,
439    domEventName: DOMEventName,
440    eventSystemFlags: EventSystemFlags,
441    isCapturePhaseListener: boolean,
442    isDeferredListenerForLegacyFBSupport?: boolean,
443  ) {
444    // 获取对应事件，事件定义在ReactDOMEventListener.js中
445    // 如DiscreteEventPriority对应dispatchDiscreteEvent，ContinuousEventPriority对应dispatchContinuousEvent
446    let listener = createEventListenerWrapperWithPriority(
447      targetContainer,
448      domEventName,
449      eventSystemFlags,
450    );
```

## createEventListenerWrapperWithPriority

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
export function createEventListenerWrapperWithPriority(
  targetContainer: EventTarget,
  domEventName: DOMEventName,
  eventSystemFlags: EventSystemFlags,
): Function {
  // 根据事件名称，获取优先级。比如click、input、drop等对应DiscreteEventPrior
  // message也许处于Scheduler中，根据getCurrentSchedulerPriorityLevel()获
  const eventPriority = getEventPriority(domEventName);
  let listenerWrapper;
```

```
  switch (eventPriority) {
    case DiscreteEventPriority:
      listenerWrapper = dispatchDiscreteEvent;
      break;
    case ContinuousEventPriority:
      listenerWrapper = dispatchContinuousEvent;
      break;
    case DefaultEventPriority:
    default:
      listenerWrapper = dispatchEvent;
      break;
  }
  return listenerWrapper.bind(
    null,
    domEventName,
    eventSystemFlags,
    targetContainer,
  );
}
```

# dispatchDiscreteEvent

## 适用事件

click、drop、input、drop 等

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
// Used by SimpleEventPlugin:
    case 'cancel':
    case 'click':
    case 'close':
    case 'contextmenu':
    case 'copy':
    case 'cut':
    case 'auxclick':
```

```
    case 'dblclick':
    case 'dragend':
    case 'dragstart':
    case 'drop':
    case 'focusin':
    case 'focusout':
    case 'input':
    case 'invalid':
    case 'keydown':
    case 'keypress':
    case 'keyup':
    case 'mousedown':
    case 'mouseup':
    case 'paste':
    case 'pause':
    case 'play':
    case 'pointercancel':
    case 'pointerdown':
    case 'pointerup':
    case 'ratechange':
    case 'reset':
    case 'resize':
    case 'seeked':
    case 'submit':
    case 'touchcancel':
    case 'touchend':
    case 'touchstart':
    case 'volumechange':
    // Used by polyfills: (fall through)
    case 'change':
    case 'selectionchange':
    case 'textInput':
    case 'compositionstart':
    case 'compositionend':
    case 'compositionupdate':
    // Only enableCreateEventHandleAPI: (fall through)
    case 'beforeblur':
    case 'afterblur':
    // Not used by React but could be by user code: (fall through)
    case 'beforeinput':
    case 'blur':
```

```
    case 'fullscreenchange':
    case 'focus':
    case 'hashchange':
    case 'popstate':
    case 'select':
    case 'selectstart':

     case 'message': {
      // We might be in the Scheduler callback.
      // Eventually this mechanism will be replaced by a check
      // of the current priority on the native scheduler.
      const schedulerPriority = getCurrentSchedulerPriorityLevel();
      switch (schedulerPriority) {
        case ImmediateSchedulerPriority:
          return DiscreteEventPriority;

        case UserBlockingSchedulerPriority:
          return ContinuousEventPriority;
        case NormalSchedulerPriority:
        case LowSchedulerPriority:
          // TODO: Handle LowSchedulerPriority, somehow. Maybe the sam
          return DefaultEventPriority;
        case IdleSchedulerPriority:
          return IdleEventPriority;
        default:
          return DefaultEventPriority;
      }
    }
```

## 派发事件源码

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
function dispatchDiscreteEvent(
  domEventName: DOMEventName,
```

```
  eventSystemFlags: EventSystemFlags,
  container: EventTarget,
  nativeEvent: AnyNativeEvent,
) {
  // ! 1. 记录上一次的事件优先级
  const previousPriority = getCurrentUpdatePriority();
  // ! 2. 记录上一次的transition
  const prevTransition = ReactCurrentBatchConfig.transition;
  // !3. 清空transition，transition为非紧急更新，这里不处理
  ReactCurrentBatchConfig.transition = null;
  try {
  // !4. 设置当前事件优先级为DiscreteEventPriority
    setCurrentUpdatePriority(DiscreteEventPriority);
    // !5. 调用dispatchEvent，执行事件
    dispatchEvent(domEventName, eventSystemFlags, container, nativeEve
  } finally {
    // !6. 恢复
    setCurrentUpdatePriority(previousPriority);
    ReactCurrentBatchConfig.transition = prevTransition;
  }
}
```

## 事件优先级记录

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

JavaScript

```
export opaque type EventPriority = Lane;

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

let currentUpdatePriority: EventPriority = NoLane;

export function getCurrentUpdatePriority(): EventPriority {
  return currentUpdatePriority;
}
```

```
export function setCurrentUpdatePriority(newPriority: EventPriority) {
  currentUpdatePriority = newPriority;
}
```

# dispatchContinuousEvent

## 适用事件

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
    case 'drag':
    case 'dragenter':
    case 'dragexit':
    case 'dragleave':
    case 'dragover':
    case 'mousemove':
    case 'mouseout':
    case 'mouseover':
    case 'pointermove':
    case 'pointerout':
    case 'pointerover':
    case 'scroll':
    case 'toggle':
    case 'touchmove':
    case 'wheel':
    // Not used by React but could be by user code: (fall through)
    case 'mouseenter':
    case 'mouseleave':
    case 'pointerenter':
    case 'pointerleave':

     case 'message': {
      // We might be in the Scheduler callback.
      // Eventually this mechanism will be replaced by a check
      // of the current priority on the native scheduler.
      const schedulerPriority = getCurrentSchedulerPriorityLevel();
      switch (schedulerPriority) {
```

```
        case ImmediateSchedulerPriority:
          return DiscreteEventPriority;

        case UserBlockingSchedulerPriority:
          return ContinuousEventPriority;


        case NormalSchedulerPriority:
        case LowSchedulerPriority:
          // TODO: Handle LowSchedulerPriority, somehow. Maybe the sam
          return DefaultEventPriority;
        case IdleSchedulerPriority:
          return IdleEventPriority;
        default:
          return DefaultEventPriority;
      }
    }
```

## 派发事件源码

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
function dispatchContinuousEvent(
  domEventName: DOMEventName,
  eventSystemFlags: EventSystemFlags,
  container: EventTarget,
  nativeEvent: AnyNativeEvent,
) {
  const previousPriority = getCurrentUpdatePriority();
  const prevTransition = ReactCurrentBatchConfig.transition;
  ReactCurrentBatchConfig.transition = null;
  try {
    setCurrentUpdatePriority(ContinuousEventPriority);
    dispatchEvent(domEventName, eventSystemFlags, container, nativeEve
  } finally {
    setCurrentUpdatePriority(previousPriority);
    ReactCurrentBatchConfig.transition = prevTransition;
```

```
  }
}
```

# dispatchEvent

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
export function dispatchEvent(
  domEventName: DOMEventName,
  eventSystemFlags: EventSystemFlags,
  targetContainer: EventTarget,
  nativeEvent: AnyNativeEvent,
): void {
  // 有些场景下是禁止事件的，比如在commit阶段
  if (!_enabled) {
    return;
  }

  let blockedOn = findInstanceBlockingEvent(nativeEvent);
  if (blockedOn === null) {
    dispatchEventForPluginEventSystem(
      domEventName,
      eventSystemFlags,
      nativeEvent,
      return_targetInst,
      targetContainer,
    );
    clearIfContinuousEvent(domEventName, nativeEvent);
    return;
  }

  if (
    queueIfContinuousEvent(
      blockedOn,
      domEventName,
      eventSystemFlags,
      targetContainer,
      nativeEvent,
```

```
    )
  ) {
    nativeEvent.stopPropagation();
    return;
  }
  // We need to clear only if we didn't queue because
  // queueing is accumulative.
  clearIfContinuousEvent(domEventName, nativeEvent);

  if (
    eventSystemFlags & IS_CAPTURE_PHASE &&
    isDiscreteEventThatRequiresHydration(domEventName)
  ) {
    while (blockedOn !== null) {
      const fiber = getInstanceFromNode(blockedOn);
      if (fiber !== null) {
        attemptSynchronousHydration(fiber);
      }
      const nextBlockedOn = findInstanceBlockingEvent(nativeEvent);
      if (nextBlockedOn === null) {
        dispatchEventForPluginEventSystem(
          domEventName,
          eventSystemFlags,
          nativeEvent,
          return_targetInst,
          targetContainer,
        );
      }
      if (nextBlockedOn === blockedOn) {
        break;
      }
      blockedOn = nextBlockedOn;
    }
    if (blockedOn !== null) {
      nativeEvent.stopPropagation();
    }
    return;
  }

  // This is not replayable so we'll invoke it but without a target,
  // in case the event system needs to trace it.
```

```
  dispatchEventForPluginEventSystem(
    domEventName,
    eventSystemFlags,
    nativeEvent,
    null,
    targetContainer,
  );
}
```

## findInstanceBlockingEvent

react/packages/react-dom-bindings/src/events/ReactDOMEventListener.js

JavaScript

```
export function findInstanceBlockingEvent(
  nativeEvent: AnyNativeEvent,
): null | Container | SuspenseInstance {
  const nativeEventTarget = getEventTarget(nativeEvent);
  return findInstanceBlockingTarget(nativeEventTarget);
}

export let return_targetInst: null | Fiber = null;

// 如果被阻塞，返回一个 SuspenseInstance 或 Container。
// 上面的 return_targetInst 字段在概念上是返回值的一部分。
export function findInstanceBlockingTarget(
  targetNode: Node,
): null | Container | SuspenseInstance {

  return_targetInst = null;

  // 通过 targetNode 获取最近的 Fiber 实例
  let targetInst = getClosestInstanceFromNode(targetNode);

  if (targetInst !== null) {
    // 寻找最近的已挂载的 Fiber 实例
    const nearestMounted = getNearestMountedFiber(targetInst);
    if (nearestMounted === null) {
```

```
      // This tree has been unmounted already. Dispatch without a targ
      // 这棵树已经被卸载了。在没有目标的情况下进行派发。
      targetInst = null;
    } else {
      const tag = nearestMounted.tag;
      if (tag === SuspenseComponent) {
        // 寻找最近的已挂载的 Suspense 实例
        const instance = getSuspenseInstanceFromFiber(nearestMounted);
        if (instance !== null) {
          // 将事件排队以便稍后重播。中止事件分发，因为我们不希望通过事件系统将此
          return instance;
        }
        // 这不应该发生，出了点问题，但为了避免阻塞整个系统，以没有目标的方式分发
        targetInst = null;
      } else if (tag === HostRoot) {
        const root: FiberRoot = nearestMounted.stateNode;
        if (isRootDehydrated(root)) {
          return getContainerFromFiber(nearestMounted);
        }
        targetInst = null;
      } else if (nearestMounted !== targetInst) {
        // 如果在提交该组件的挂载之前收到事件（例如：图片加载完成），暂时忽略它（
        // 我们也可以考虑将事件排队，并在挂载后分发它们。
        targetInst = null;
      }
    }
  }
  return_targetInst = targetInst;
  // 没有阻塞
  return null;
}
```

## dispatchEventForPluginEventSystem

react/packages/react-dom-bindings/src/events/DOMPluginEventSystem.js

JavaScript

```
export function dispatchEventForPluginEventSystem(
  domEventName: DOMEventName,
  eventSystemFlags: EventSystemFlags,
```

```
  nativeEvent: AnyNativeEvent,
  targetInst: null | Fiber,
  targetContainer: EventTarget,
): void {
  let ancestorInst = targetInst;
  if (
    (eventSystemFlags & IS_EVENT_HANDLE_NON_MANAGED_NODE) === 0 &&
    (eventSystemFlags & IS_NON_DELEGATED) === 0
  ) {
    const targetContainerNode = ((targetContainer: any): Node);

    if (targetInst !== null) {
      let node: null | Fiber = targetInst;

      mainLoop: while (true) {
        if (node === null) {
          // 事件没有对应的fiber，没法执行事件，退出
          return;
        }
        const nodeTag = node.tag;
        if (nodeTag === HostRoot || nodeTag === HostPortal) {
          let container = node.stateNode.containerInfo;
          if (isMatchingRootContainer(container, targetContainerNode))
            // container和targetContainerNode相等，说明找到了对应的rootCon
            break;
          }
          if (nodeTag === HostPortal) {
            // 代码略...
          }
          // 现在我们需要在另一棵树中找到它对应的宿主 fiber。为此，我们可以使用
          // 但我们需要验证该 fiber 是否是宿主实例，否则我们需要沿着 DOM 向上遍
         // 代码略...
        }
        node = node.return;
      }
    }
  }

  // 批量更新
  batchedUpdates(() =>
    dispatchEventsForPlugins(
```

```
      domEventName,
      eventSystemFlags,
      nativeEvent,
      ancestorInst,
      targetContainer,
    ),
  );
}
```

## 批量更新 batchedUpdates

react/packages/react-dom-bindings/src/events/ReactDOMUpdateBatching.js

JavaScript

```
export function batchedUpdates(fn, a, b) {
  if (isInsideEventHandler) {
    // 如果我们当前正在另一个批处理中，需要等待其完全完成后再恢复状态。
    return fn(a, b);
  }
  isInsideEventHandler = true;
  try {
    return batchedUpdatesImpl(fn, a, b);
  } finally {
    isInsideEventHandler = false;
    finishEventHandler();
  }
}
```

batchedUpdatesImpl 的具体实现在 react/packages/react-reconciler/src/ReactFiberWorkLoop.js

JavaScript

```
export function batchedUpdates<A, R>(fn: A => R, a: A): R {
  const prevExecutionContext = executionContext;
  executionContext |= BatchedContext;
  try {
    return fn(a);
  } finally {
    executionContext = prevExecutionContext;
```

```
  }
}
```

## dispatchEventsForPlugins

JavaScript

```
function dispatchEventsForPlugins(
  domEventName: DOMEventName,
  eventSystemFlags: EventSystemFlags,
  nativeEvent: AnyNativeEvent,
  targetInst: null | Fiber,
  targetContainer: EventTarget,
): void {
  // nativeEvent.target
  const nativeEventTarget = getEventTarget(nativeEvent);
  const dispatchQueue: DispatchQueue = [];
  extractEvents(
    dispatchQueue,
    domEventName,
    targetInst,
    nativeEvent,
    nativeEventTarget,
    eventSystemFlags,
    targetContainer,
  );
  processDispatchQueue(dispatchQueue, eventSystemFlags);
}
```

### extractEvents

获取事件对象，如 click 事件对应的 SyntheticMouseEvent。

DebugReact > src > react > packages > react-dom-bindings > src > events > plugins > SimpleEventPlugin.js > extractEvents

```
97      case 'click':
98        // Firefox creates a click event on right mouse clicks. This removes the
99        // unwanted click events.
100       // TODO: Fixed in https://phabricator.services.mozilla.com/D26793. Can
101       // probably remove.
102       if (nativeEvent.button === 2) {
103         return;
104       }
105     /* falls through */
106     case 'auxclick':
107     case 'dblclick':
108     case 'mousedown':
109     case 'mousemove':
110     case 'mouseup':
111     // TODO: Disabled elements should not respond to mouse events
112     /* falls through */
113     case 'mouseout':
114     case 'mouseover':
115     case 'contextmenu':
116       SyntheticEventCtor = SyntheticMouseEvent;
117       break;
```

## processDispatchQueue

JavaScript

```
export function processDispatchQueue(
  dispatchQueue: DispatchQueue,
  eventSystemFlags: EventSystemFlags,
): void {
  const inCapturePhase = (eventSystemFlags & IS_CAPTURE_PHASE) !== 0;
  for (let i = 0; i < dispatchQueue.length; i++) {
    const {event, listeners} = dispatchQueue[i];
    processDispatchQueueItemsInOrder(event, listeners, inCapturePhase)
  }
  // throw error
  rethrowCaughtError();
}
function processDispatchQueueItemsInOrder(
  event: ReactSyntheticEvent,
  dispatchListeners: Array<DispatchListener>,
  inCapturePhase: boolean,
): void {
  let previousInstance;
  if (inCapturePhase) {
    for (let i = dispatchListeners.length - 1; i >= 0; i--) {
      const {instance, currentTarget, listener} = dispatchListeners[i]
      if (instance !== previousInstance && event.isPropagationStopped(
        return;
```

```
      }
      // 执行事件
      executeDispatch(event, listener, currentTarget);
      previousInstance = instance;
    }
  } else {
    for (let i = 0; i < dispatchListeners.length; i++) {
      const {instance, currentTarget, listener} = dispatchListeners[i]
      if (instance !== previousInstance && event.isPropagationStopped(
        return;
      }
      // 执行事件
      executeDispatch(event, listener, currentTarget);
      previousInstance = instance;
    }
  }
}
```

**执行事件**

JavaScript

```
function executeDispatch(
  event: ReactSyntheticEvent,
  listener: Function,
  currentTarget: EventTarget,
): void {
  const type = event.type || 'unknown-event';
  event.currentTarget = currentTarget;
  invokeGuardedCallbackAndCatchFirstError(type, listener, undefined, e
  event.currentTarget = null;
}
```

react/packages/shared/ReactErrorUtils.js

JavaScript

```
export function invokeGuardedCallbackAndCatchFirstError<
  A,
  B,
```

```
  C,
  D,
  E,
  F,
  Context,
>(
  this: mixed,
  name: string | null,
  func: (a: A, b: B, c: C, d: D, e: E, f: F) => void,
  context: Context,
  a: A,
  b: B,
  c: C,
  d: D,
  e: E,
  f: F,
): void {
  invokeGuardedCallback.apply(this, arguments);
  if (hasError) {
    const error = clearCaughtError();
    if (!hasRethrowError) {
      hasRethrowError = true;
      rethrowError = error;
    }
  }
}

export function invokeGuardedCallback<A, B, C, D, E, F, Context>(
  name: string | null,
  func: (a: A, b: B, c: C, d: D, e: E, f: F) => mixed,
  context: Context,
  a: A,
  b: B,
  c: C,
  d: D,
  e: E,
  f: F,
): void {
  hasError = false;
  caughtError = null;
```

```
  invokeGuardedCallbackImpl.apply(reporter, arguments);
}
```

react/packages/shared/invokeGuardedCallbackImpl.js

JavaScript

```
export default function invokeGuardedCallbackImpl<Args: Array<mixed>,
  this: {onError: (error: mixed) => void},
  name: string | null,
  func: (...Args) => mixed,
  context: Context,
): void {
   const funcArgs = Array.prototype.slice.call(arguments, 3);
    try {
      func.apply(context, funcArgs);
    } catch (error) {
      this.onError(error);
    }
}
```