深入了解现代网络浏览器（第 4 部分）

Mariko Kosaka

输入即将到达合成器

这是深入探究 Chrome 的 4 篇博文系列的最后一篇，我们将探究 Chrome 如何处理我们的代码以显示网站。在上一篇博文中，我们介绍了渲染流程并了解了合成器。在本文中，我们将了解合成器如何在有用户输入时实现流畅的互动。

从浏览器的角度看输入事件

当您听到“输入事件”时，可能只会想到在文本框中输入内容或鼠标点击，但从浏览器的角度来看，输入是指用户的任何手势。鼠标滚轮滚动是一种输入事件，触摸或鼠标悬停也是一种输入事件。

当屏幕上发生用户手势（例如轻触）时，浏览器进程是首先接收手势的进程。但是，由于标签页中的内容由渲染器进程处理，因此浏览器进程只知道该手势发生的位置。因此，浏览器进程会将事件类型（例如 `touchstart`）及其坐标发送给渲染程序进程。渲染程序进程会通过查找事件目标并运行已附加的事件监听器来妥善处理事件。

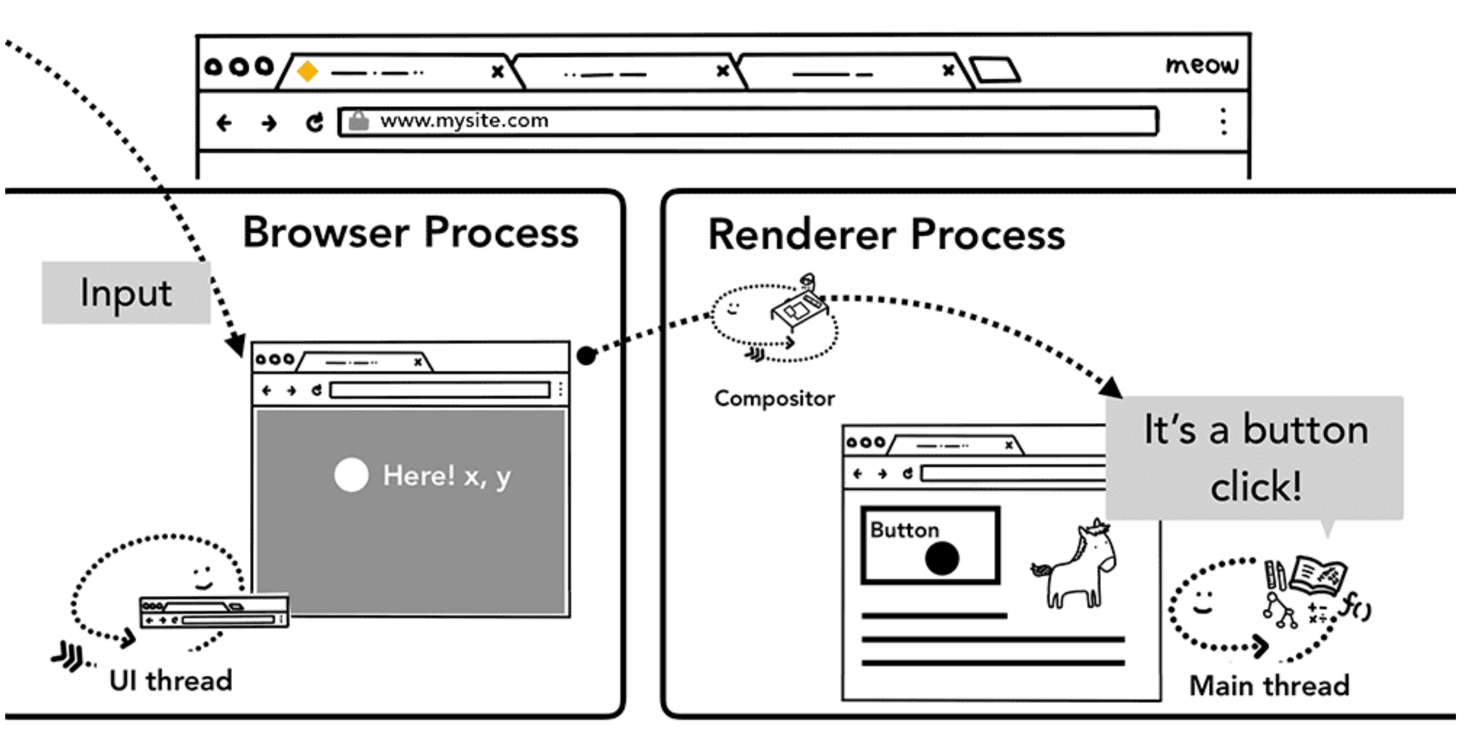

图 1：输入事件通过浏览器进程路由到渲染程序

合成器接收输入事件

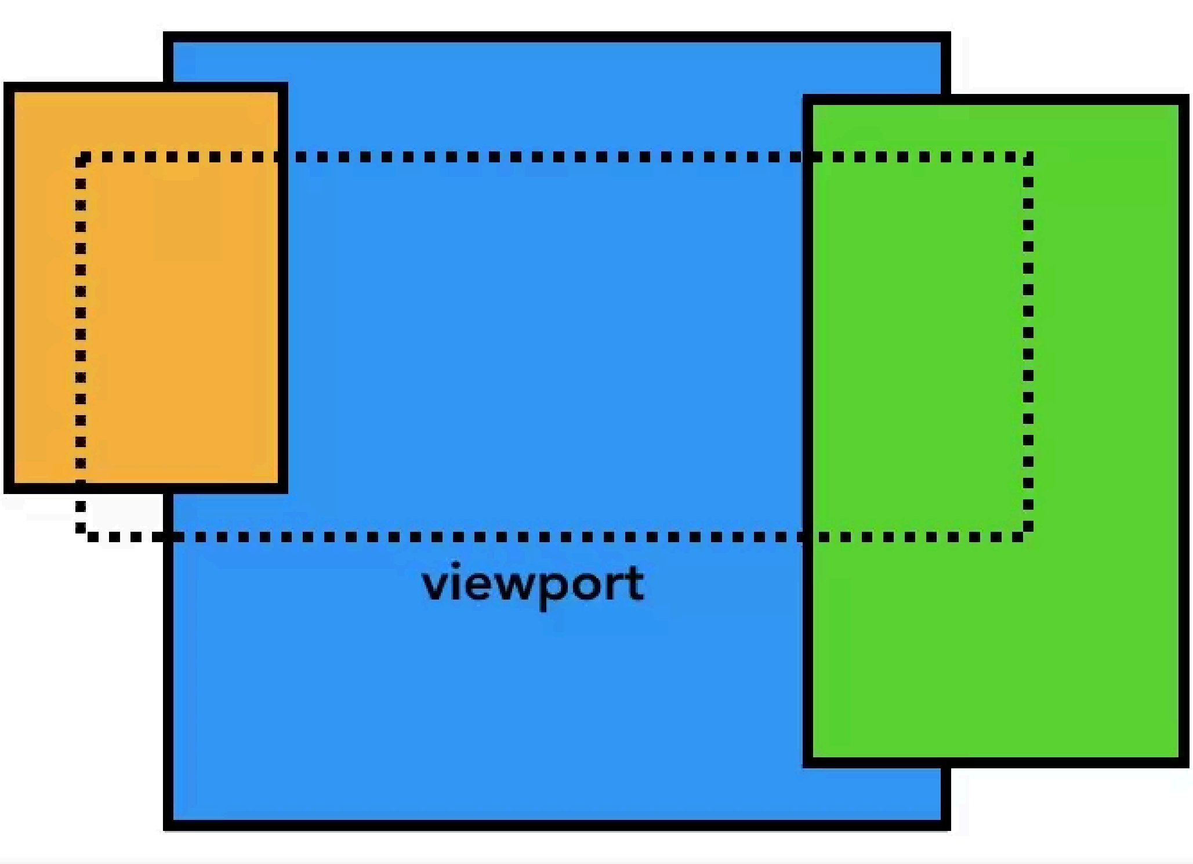

图 2：视口悬停在页面层上

在上一篇博文中，我们介绍了合成光栅化图层后，合成器如何流畅地处理滚动。如果未将任何输入事件监听器附加到页面，则合成器线程可以创建与主线程完全独立的新合成帧。但是，如果网页上附加了一些事件监听器，该怎么办？合成器线程如何确定是否需要处理事件？

了解非快速可滚动区域

由于运行 JavaScript 是主线程的工作，因此在合成网页时，合成器线程会将网页中附加了事件处理脚本的区域标记为“非快速滚动区域”。有了这些信息，如果事件发生在该区域，合成器线程可以确保将输入事件发送到主线程。如果输入事件来自此区域之外，则合成器线程会继续合成新帧，而无需等待主线程。

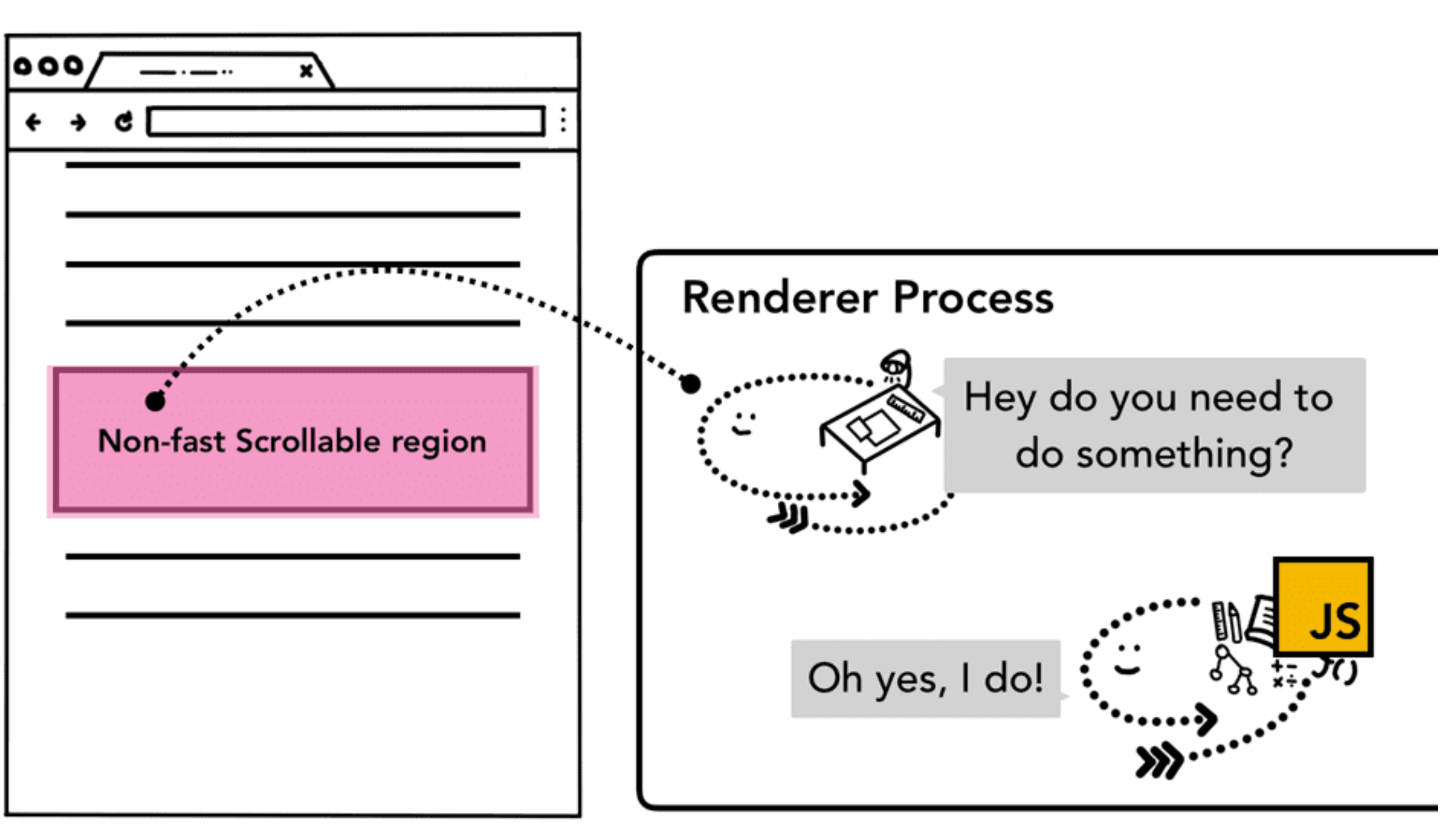

图 3：对非快速滚动区域的描述输入的示意图

编写事件处理程序时须注意

在 Web 开发中，事件委托是一种常见的事件处理模式。由于事件会向上冒泡，因此您可以在最顶部的元素上附加一个事件处理脚本，并根据事件目标委托任务。您可能看过或编写过如下代码。

```
document.body.addEventListener('touchstart', event => {
    if (event.target === area) {
        event.preventDefault();
    }
});
```

由于您只需为所有元素编写一个事件处理脚本，因此这种事件委托模式的人体工学设计非常吸引人。不过，如果您从浏览器的角度查看此代码，现在整个网页都被标记为非快速滚动区域。这意味着，即使您的应用不关心来自网页某些部分的输入，每当有输入事件传入时，合成器线程都必须与主线程通信并等待它。因此，合成器的流畅滚动功能会失效。

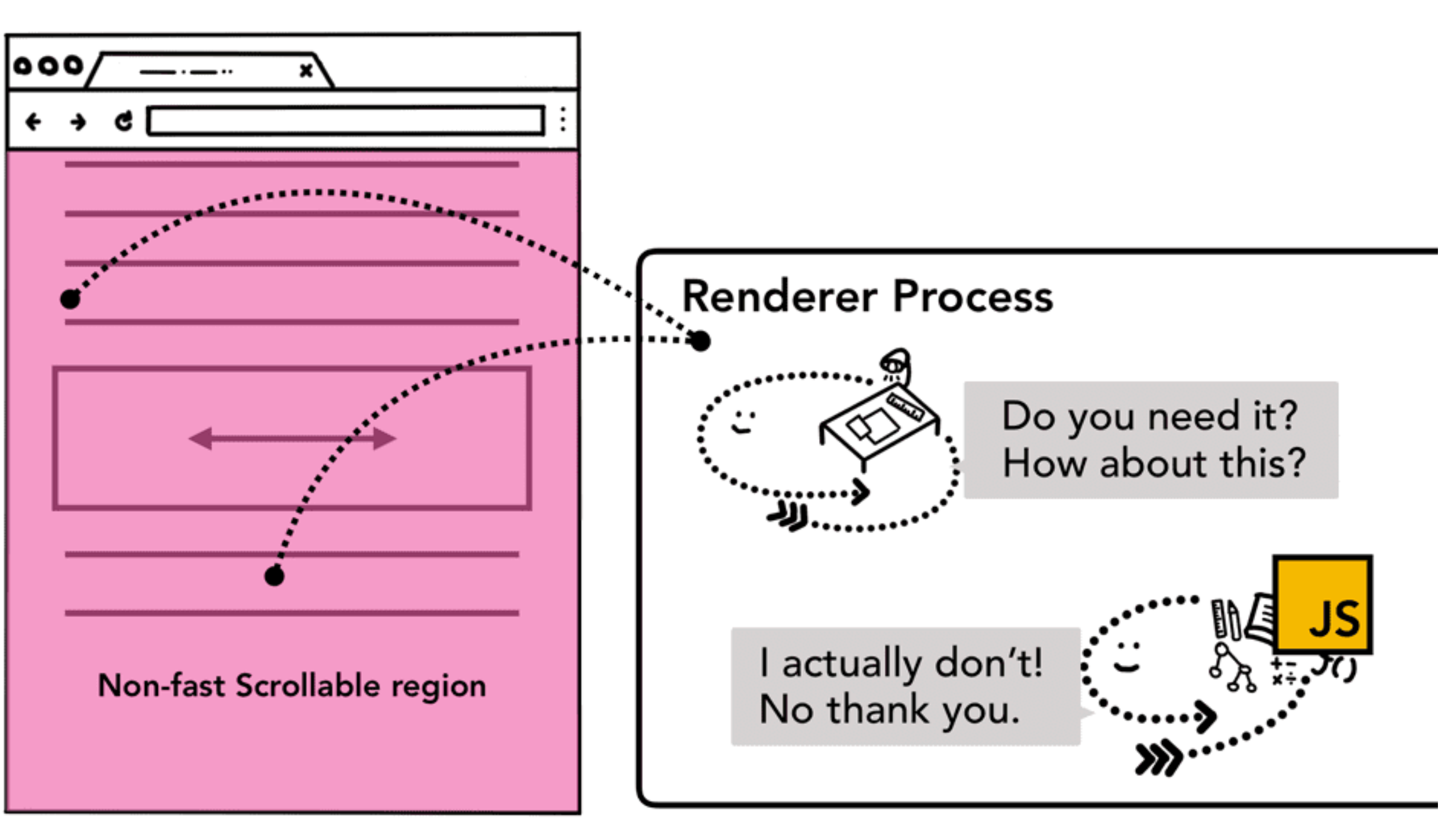

图 4：对覆盖整个页面的非快速滚动区域的描述输入的示意图

为避免出现这种情况，您可以在事件监听器中传递 `passive: true` 选项。这会提示浏览器您仍希望在主线程中监听事件，但合成器也可以继续合成新帧。

```
document.body.addEventListener('touchstart', event => {
    if (event.target === area) {
        event.preventDefault()
    }
 }, {passive: true});
```

检查事件是否可取消

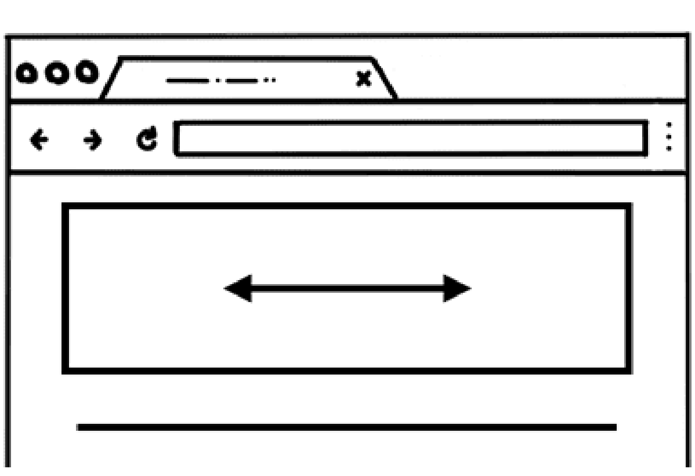

图 5：网页的部分内容固定为水平滚动

假设您在页面中有一个框，并且希望将滚动方向限制为仅限水平滚动。

在指针事件中使用 `passive: true` 选项意味着网页滚动可以是流畅的，但在您想要 `preventDefault` 以限制滚动方向时，垂直滚动可能已开始。您可以使用 `event.cancelable` 方法进行检查。

```
document.body.addEventListener('pointermove', event => {
    if (event.cancelable) {
        event.preventDefault(); // block the native scroll
        /*
        *  do what you want the application to do here
        */
    }
}, {passive: true});
```

或者，您也可以使用 `touch-action` 等 CSS 规则来彻底消除事件处理脚本。

```
#area {
  touch-action: pan-x;
}
```

查找事件目标

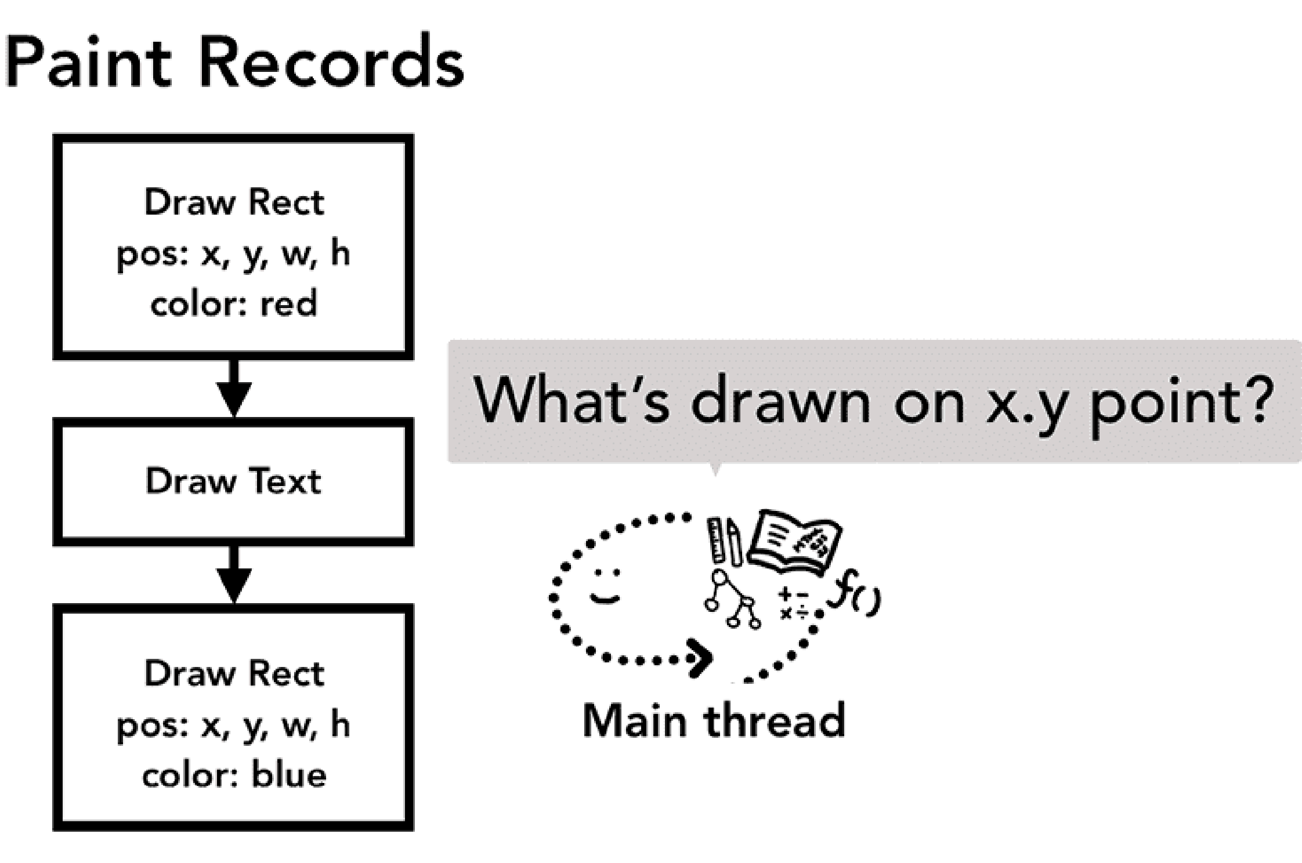

图 6：主线程查看绘制记录，询问在 x.y 点绘制了什么

当合成器线程向主线程发送输入事件时，首先要运行一个命中测试来查找事件目标。点击测试使用渲染过程中生成的绘制记录数据，以了解事件发生点坐标下方的内容。

尽量减少对主线程的事件调度

在上一篇博文中，我们讨论了典型显示屏如何每秒刷新屏幕 60 次，以及我们如何需要跟上节奏以实现流畅的动画。对于输入，典型的触摸屏设备每秒可传送 60-120 次轻触事件，典型的鼠标每秒可传送 100 次事件。输入事件的保真度高于屏幕刷新的速度。

如果每秒向主线程发送 120 次 `touchmove` 等连续事件，那么相对于屏幕刷新速度，它可能会触发过多的点击测试和 JavaScript 执行。

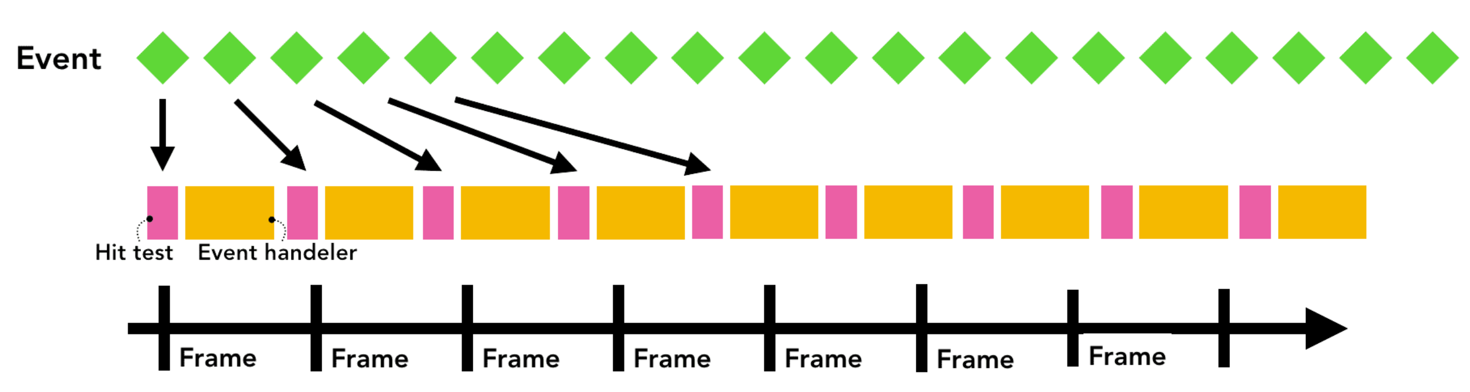

图 7：事件充斥帧时间轴，导致页面卡顿

为了尽量减少对主线程的过多调用，Chrome 会合并连续事件（例如 `wheel`、`mousewheel`、`mousemove`、`pointermove`、`touchmove`），并将调度延迟到下一个 `requestAnimationFrame` 之前。

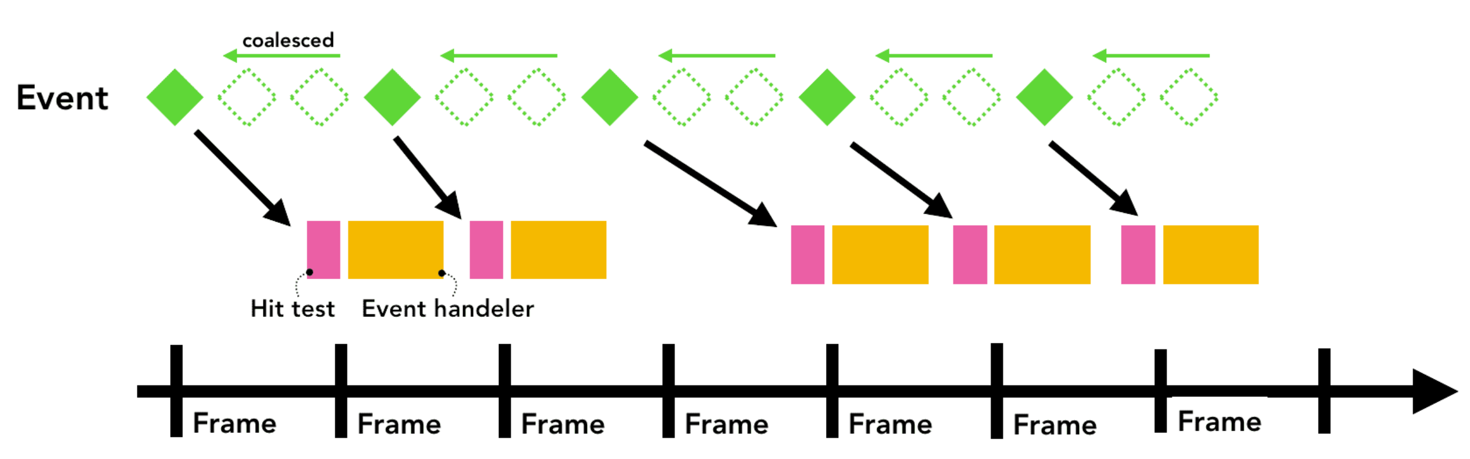

图 8：与之前相同的时间轴，但事件被合并并延迟了

系统会立即调度任何离散事件，例如 `keydown`、`keyup`、`mouseup`、`mousedown`、`touchstart` 和 `touchend`。

使用 `getCoalescedEvents` 获取帧内事件

对于大多数 Web 应用来说，合并事件应该足以提供良好的用户体验。但是，如果您要构建绘制应用等内容并根据 `touchmove` 坐标放置路径，则可能会丢失绘制平滑线条所需的中间坐标。在这种情况下，您可以在指针事件中使用 `getCoalescedEvents` 方法来获取有关这些合并事件的信息。

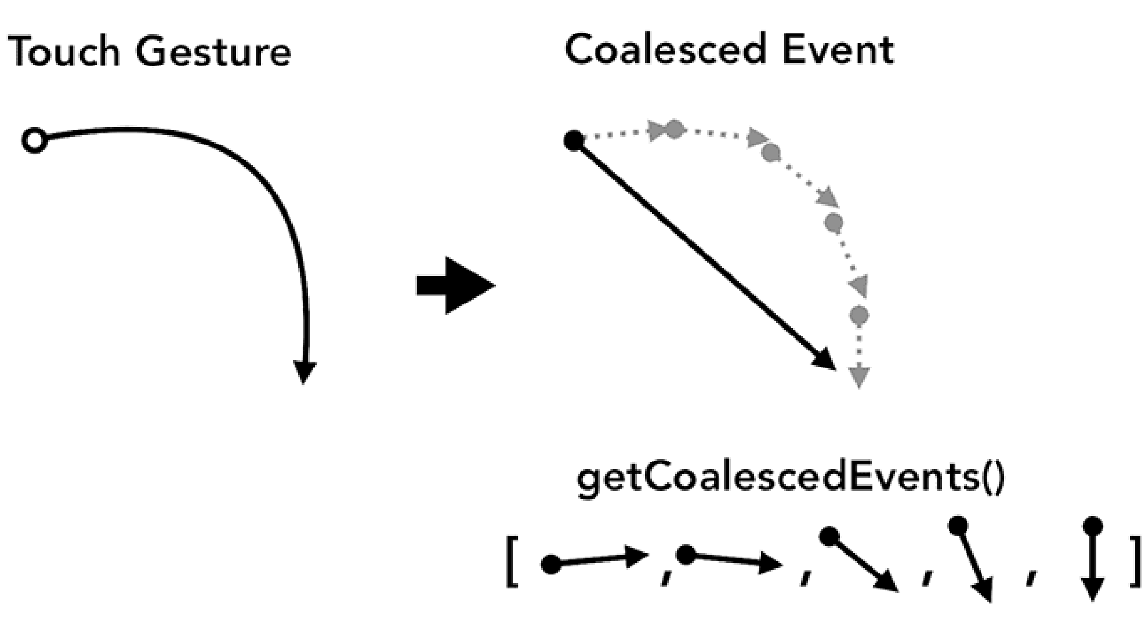

图 9：左侧是平滑的触摸手势路径，右侧是合并的受限路径

```
window.addEventListener('pointermove', event => {
    const events = event.getCoalescedEvents();
    for (let event of events) {
        const x = event.pageX;
        const y = event.pageY;
        // draw a line using x and y coordinates.
    }
});
```

后续步骤

在本系列文章中，我们介绍了网络浏览器的内部运作方式。如果您从未想过为什么 DevTools 建议在事件处理脚本中添加 `{passive: true}`，或者为什么您可能需要在脚本标记中编写 `async` 属性，希望本系列文章能让您了解浏览器为何需要这些信息来提供更快、更流畅的 Web 体验。

使用 Lighthouse

如果您希望自己的代码对浏览器友好，但不知道从何入手，不妨使用 Lighthouse 这款工具来对任何网站运行审核，并生成报告，指出哪些做法正确、哪些需要改进。仔细阅读审核列表，您还可以了解浏览器关注哪些方面。

了解如何衡量效果

效果调整可能因网站而异，因此请务必衡量您网站的效果，并确定最适合您网站的调整。Chrome DevTools 团队提供了一些有关如何衡量网站性能的教程。

在您的网站上添加功能政策

如果您想采取额外措施，功能政策是一项新的 Web 平台功能，可在您构建项目时为您提供防护。启用功能政策可保证应用的特定行为，并防止您出错。例如，如果您想确保应用绝不会阻止解析，可以采用同步脚本政策来运行应用。启用 `sync-script: 'none'` 后，系统会阻止执行解析器阻塞型 JavaScript。这样可以防止任何代码阻塞解析器，浏览器也不必担心暂停解析器。

小结

刚开始构建网站时，我几乎只关心如何编写代码以及如何提高工作效率。这些方面很重要，但我们还应考虑浏览器如何处理我们编写的代码。现代浏览器一直在不断投资，以便为用户提供更好的网络体验。通过整理代码来为浏览器提供良好的体验，进而改善用户体验。希望您能加入我们，共同为善待浏览器而努力！

非常感谢审核本系列图书早期草稿的所有人，包括（但不限于）：Alex Russell、Paul Irish、Meggin Kearney、Eric Bidelman、Mathias Bynens、Addy Osmani、Kinuko Yasuda、Nasko Oskov 和 Charlie Reis。

您喜欢本系列文章吗？如果您对日后发布的博文有任何疑问或建议，欢迎在下方的评论区留言，或在 Twitter 上通过 @kosamari 与我联系。