Making touch scrolling fast by default

By Dave Tapuska

Dave is a contributor to Web**Fundamentals**

We know that scrolling responsiveness is critical to the user's engagement with a website on mobile, yet touch event listeners often cause serious scrolling performance problems. Chrome has been addressing this by allowing touch event listeners to be passive (passing the `{passive: true}` option to `addEventListener()`) and shipping the pointer events API. These are great features to drive new content into models that don't block scrolling, but developers sometimes find them hard to understand and adopt.

We believe the web should be fast by default without developers needing to understand arcane details of browser behavior. In Chrome 56 we are defaulting touch listeners to passive by default in cases where that most often matches the developer's intention. We believe that by doing this we can greatly improve the user's experience whilst having minimal negative impact on sites.

In rare cases this change can result in unintended scrolling. This is usually easily addressed by applying a touch-action: **none** style to the element where scrolling shouldn't occur. Read on for details, how to know if you are impacted, and what you can do about it.

Background: Cancelable Events slow your page down

If you call preventDefault() in the `touchstart` or first `touchmove` events then you will prevent scrolling. The problem is that most often listeners will not call `preventDefault()`, but the browser needs to wait for the event to finish to be sure of that. Developer-defined "passive event listeners" solve this. When you add a touch event with a `{passive: true}` object as the third parameter in your event handler then you are telling the browser that the

touchstart listener will not call **preventDefault()** and the browser can safely perform the scroll without blocking on the listener. For example:

```
window.addEventListener("touchstart", func, {passive: true} );
```

The Intervention

Our main motivation is to reduce the time it takes to update the display after the user touches the screen. To understand the usage of touchstart and touchmove we added metrics to determine how frequently scroll blocking behaviour occurred.

We looked at the percentage of cancelable touch events that were sent to a root target (window, document, or body) and determined that about 80% of these listeners are conceptually passive but were not registered as such. Given the scale of this problem we noticed a great opportunity to improve scrolling without any developer action by making these events automatically "passive".

This drove us to define our intervention as: if the target of a touchstart or touchmove listener is the **window**, **document** or **body** we default **passive** to **true**. This means that code like:

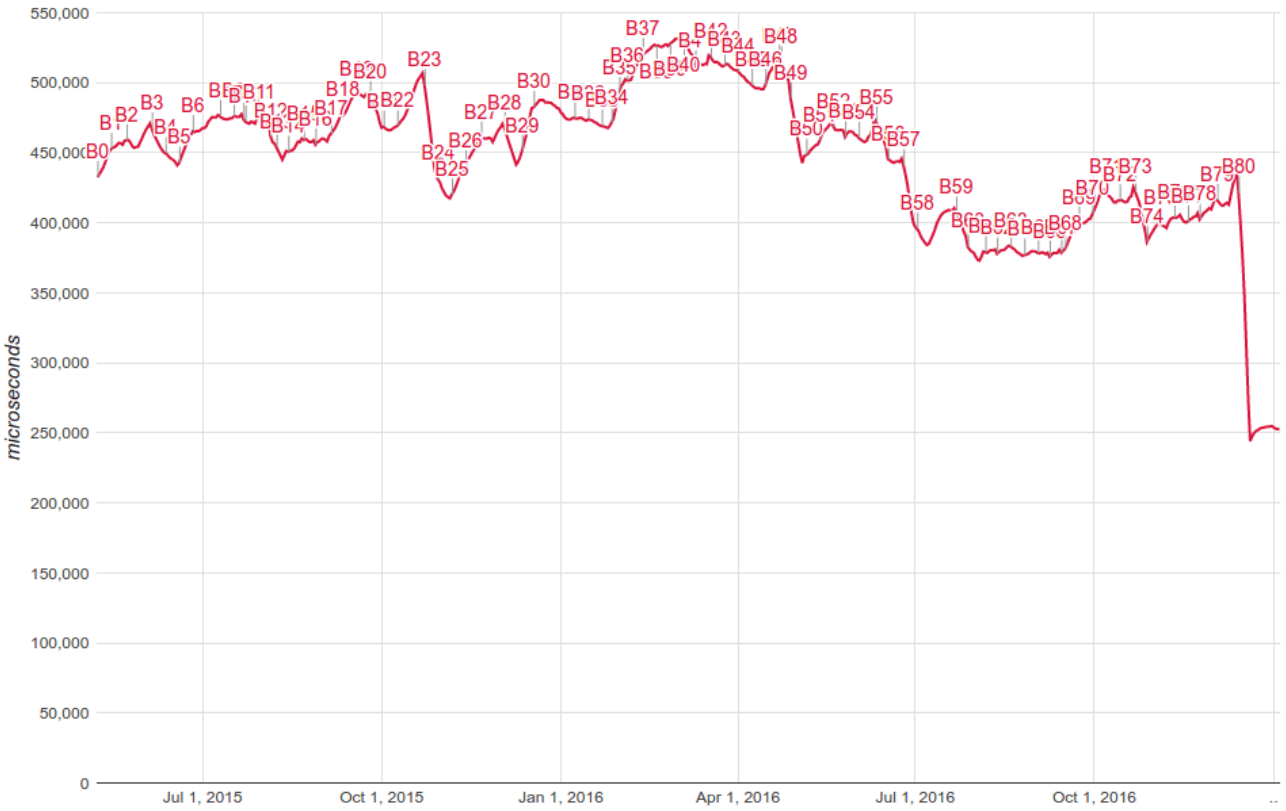
```
window.addEventListener("touchstart", func);
```

becomes equivalent to:

```
window.addEventListener("touchstart", func, {passive: true} );
```

Now calls to **preventDefault()** inside the listener will be ignored.

The graph below shows the time taken by the top 1% of scrolls from the time a user touches the screen to scroll to the time the display is updated. This data is for all websites in Chrome for Android. Before the intervention was enabled 1% of scrolls took just over 400ms. That has now been reduced to just over 250ms in Chrome 56 Beta; a reduction of about 38%. In the future we hope to make passive true the default for *all* **touchstart** and **touchmove** listeners, reducing this to below 50ms.



Breakage and Guidance

In the vast majority of cases, no breakage will be observed. But when breakage does occur, the most common symptom is that scrolling happens when you don't want it. In rare cases, developers may also notice unexpected **click** events (when `preventDefault()` was missing from a `touchend` listener).

In Chrome 56 and later, DevTools will log a warning when you call `preventDefault()` in an event where the intervention is active.

```
touch-passive.html:19 Unable to preventDefault inside passive event listene
```

Your application can determine whether it may be hitting this in the wild by checking if calling `preventDefault` had any effect via the `defaultPrevented` property.

We've found that a large majority of impacted pages are fixed relatively easily by applying the touch-action CSS property whenever possible. If you wish to prevent all browser scrolling and zooming within an element apply `touch-action: none` to it. If you have a horizontal carousel consider applying `touch-action: pan-y pinch-zoom` to it so that the user can still scroll vertically and zoom as normal. Applying touch-action correctly is already necessary on browsers such as desktop Edge that support Pointer Events and not Touch Events. For

mobile Safari and older mobile browsers that don't support touch-action your touch listeners must continue calling `preventDefault` even when it will be ignored by Chrome.

In more complex cases it may be necessary to also rely on one of the following:

- If your `touchstart` listener calls `preventDefault()`, ensure preventDefault() is also called from associated touchend listeners to continue suppressing the generation of click events and other default tap behavior.
- Last (and discouraged) pass `{passive: false}` to addEventListener() to override the default behavior. Be aware you will have to feature detect if the User Agent supports EventListenerOptions.

Conclusion

In Chrome 56 scrolling starts substantially faster on many websites. This is the only impact that most developers will notice as a result of this change. In some cases developers may notice unintended scrolling.

Although it's still necessary to do so for mobile Safari, websites should not rely on calling `preventDefault()` inside of `touchstart` and `touchmove` listeners as this is no longer guaranteed to be honored in Chrome. Developers should apply the `touch-action` CSS property on elements where scrolling and zooming should be disabled to notify the browser before any touch events occur. To suppress the default behavior of a tap (such as the generation of a click event), call `preventDefault()` inside of a `touchend` listener.

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code samples are licensed under the Apache 2.0 License. For details, see our Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.