

JAVASCRIPT FRONTEND

AVANZADO

CLASE 11 : PERSISTENCIA Y OPTIMIZACIÓN

PERSISTENCIA DE DATOS

Hasta ahora teníamos que emplear cookies u otras soluciones que usaban plugins instalados en el navegador: JRE + applets, Flash player + LSO(Local Shared Objects), SQLite + Firefox , Google Gears, ... para almacenar información persistente en el navegador cliente.

Con HTML5 contamos con una nueva técnica para almacenar datos persistentes en el navegador cliente. Cuando HTML5 esté ampliamente extendido podremos hacer uso del API DOM Storage de HTML5 con el cual los datos persistirán entre sesiones sin tener que usar las limitadas cookies (4KB por cookie). Con el API DOM Storage y los métodos `JSON.stringify` y `JSON.parse` podremos almacenar objetos complejos de un modo muy sencillo.

STORAGE API

La interfaz de almacenamiento de la API de almacenamiento web proporciona acceso al almacenamiento de la sesión o al almacenamiento local para un dominio particular, lo que le permite, por ejemplo, agregar, modificar o eliminar elementos de datos almacenados.

Si desea manipular el almacenamiento de sesión para un dominio, llame al método `Window.sessionStorage`; Si desea manipular el almacenamiento local de un dominio, llame a `Window.localStorage` ¹.

sessionStorage

La propiedad `sessionStorage` le permite acceder a un objeto de almacenamiento de sesión para el origen actual. `sessionStorage` es similar a `Window.localStorage`; la única diferencia es que mientras los datos almacenados en `localStorage` no tienen un conjunto de

caducidad, los datos almacenados en `sessionStorage` se borran cuando finaliza la sesión de la página. Una sesión de página dura todo el tiempo que el navegador está abierto y sobrevive a las recargas y restauraciones de página. Abrir una página en una nueva pestaña o ventana provocará que se inicie una nueva sesión, que difiere de cómo funcionan las cookies de sesión ³.

localStorage

La propiedad `localStorage` de solo lectura le permite acceder a un objeto de almacenamiento para el origen del documento; los datos almacenados se guardan en las sesiones del navegador.

Cabe señalar que los datos almacenados en `localStorage` o `sessionStorage` son específicos del protocolo de la página.

Los objetos de almacenamiento son simples almacenes de clave-valor, similares a los objetos, pero se mantienen intactos a través de las cargas de página. Las claves y los valores son siempre cadenas (tenga en cuenta que las claves enteras se convertirán automáticamente en cadenas, al igual que lo hacen los objetos). Puede acceder a estos valores como un objeto o con los métodos `Storage.getItem()` y `Storage.setItem()`. Estas tres líneas establecen la entrada `colorSetting` de la misma manera ²:

```
localStorage.colorSetting = '#a4509b';  
localStorage['colorSetting'] = '#a4509b';  
localStorage.setItem('colorSetting', '#a4509b');
```

setItem

El método `setItem()` de la interfaz de Almacenamiento, al pasar un nombre y valor clave, agregará esa clave al almacenamiento o actualizará el valor de esa clave si ya existe.

```
storage.setItem(keyName, keyValue);
```

`setItem()` puede arrojar una excepción si el almacenamiento está lleno. Particularmente, en Mobile Safari (desde iOS 5) siempre se lanza cuando el usuario ingresa en modo privado (Safari establece la cuota en 0 bytes en modo privado, a diferencia de otros navegadores,

que permiten el almacenamiento en modo privado, usando contenedores de datos separados). Por lo tanto, los desarrolladores deben asegurarse de capturar siempre las posibles excepciones de `setItem()` ⁴.

getItem

El método `getItem()` de la interfaz de almacenamiento, cuando se pasa un nombre de clave, devolverá el valor de esa clave ⁵.

```
var aValue = storage.getItem(keyName);
```

PERFORMANCE

RAIL es un modelo de rendimiento centrado en el usuario que descompone la experiencia del usuario en acciones clave. Los objetivos y las directrices de RAIL tienen como objetivo ayudar a los desarrolladores y diseñadores a garantizar una buena experiencia de usuario para cada una de estas acciones. Al diseñar una estructura para pensar en el rendimiento, RAIL permite a los diseñadores y desarrolladores enfocarse de manera confiable en el trabajo que tiene el mayor impacto en la experiencia del usuario ⁶.

Cada aplicación web tiene cuatro aspectos distintos a su ciclo de vida, y el rendimiento se ajusta a ellos de diferentes maneras:



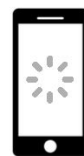
Response



Animation



Idle



Load

En el contexto de RAIL, los términos objetivos y directrices tienen significados específicos:

- Metas (Goals) : Métricas de rendimiento clave relacionadas con la experiencia del usuario. Dado que la percepción humana es relativamente constante, es poco probable que estos objetivos cambien en el corto plazo.
- Pautas (Guidelines) : Recomendaciones que lo ayudan a alcanzar los objetivos. Estos pueden ser específicos del hardware actual y las condiciones de conexión de red, y por lo tanto pueden cambiar con el tiempo.

Convierta a los usuarios en el punto focal de su esfuerzo de rendimiento. La siguiente tabla describe las métricas clave de cómo los usuarios perciben los retrasos en el rendimiento :

0 to 16ms	Los usuarios son excepcionalmente buenos en el seguimiento del movimiento y no les gusta cuando las animaciones no son suaves. Perciben las animaciones como uniformes siempre que se representen 60 nuevos fotogramas por segundo. Eso es 16 ms por cuadro, incluido el tiempo que le toma al navegador pintar el nuevo marco en la pantalla, dejando una aplicación de aproximadamente 10 ms para producir un marco.
0 to 100ms	Responda a las acciones de los usuarios dentro de esta ventana de tiempo y los usuarios sienten que el resultado es inmediato. Por más tiempo, y la conexión entre acción y reacción se rompe.
100 to 300ms	Los usuarios experimentan un ligero retraso perceptible.
300 to 1000ms	Dentro de esta ventana, las cosas se sienten parte de una progresión natural y continua de tareas. Para la mayoría de los usuarios en la

	web, cargar páginas o cambiar vistas representa una tarea.
1000ms o más	Más allá de 1000 milisegundos (1 segundo), los usuarios pierden el foco en la tarea que están realizando.

- **Respuesta: responder en menos de 50 ms**

Objetivo: completar una transición iniciada por la entrada del usuario dentro de los 100 ms. Los usuarios pasan la mayor parte del tiempo esperando que los sitios respondan a sus comentarios, sin esperar a que los sitios se carguen.

Pautas:

1. Responda a la entrada del usuario dentro de los 50 ms o, de lo contrario, la conexión entre la acción y la reacción se interrumpe. Esto se aplica a la mayoría de las entradas, como hacer clic en los botones, alternar controles de formulario o iniciar animaciones. Esto no se aplica a roces táctiles o pergaminos.
2. Aunque puede parecer contradictorio, no siempre es la decisión correcta para responder a los comentarios del usuario de inmediato. Puede usar esta ventana de 100 ms para hacer otro trabajo costoso. Pero ten cuidado de no bloquear al usuario. Si es posible, trabaje en segundo plano.
3. Para realizar acciones que tarden más de 50 ms en completarse, siempre brinde comentarios.

- **Animación: produce un marco en 10ms**

Metas: Produzca cada cuadro en una animación en 10ms o menos.

Técnicamente, el presupuesto máximo para cada fotograma es de 16 ms (1000 ms / 60 fotogramas por segundo \approx 16 ms), pero los navegadores necesitan aproximadamente 6 ms para representar cada fotograma, por lo tanto, la pauta de 10 ms por fotograma. Apunta a la suavidad visual. Los usuarios notan cuando las tasas de cuadros varían.

Pautas:

1. En puntos de alta presión como animaciones, la clave es no hacer nada donde se pueda, y el mínimo absoluto donde no se pueda. Siempre que sea posible, utilice la

respuesta de 100 ms para calcular previamente el trabajo costoso para que pueda maximizar sus posibilidades de alcanzar los 60 fps.

2. Consulte Rendering Performance para varias estrategias de optimización de animación.
3. Reconoce todos los tipos de animaciones. Las animaciones no son solo efectos de interfaz de usuario de lujo. Cada una de estas interacciones se consideran animaciones:
 - a. Animaciones visuales, como entradas y salidas, interpolaciones e indicadores de carga.
 - b. Desplazamiento(scrolling). Esto incluye flinging, que es cuando el usuario comienza a desplazarse, luego suelta y la página continúa desplazándose.
 - c. Arrastrando(Dragging) .Las animaciones suelen seguir las interacciones del usuario, como desplazarse por un mapa o pellizcar para hacer zoom.

● Inactivo: maximiza el tiempo de inactividad

Objetivo: Maximizar el tiempo de inactividad para aumentar las probabilidades de que la página responda a la entrada del usuario dentro de los 50 ms.

Pautas:

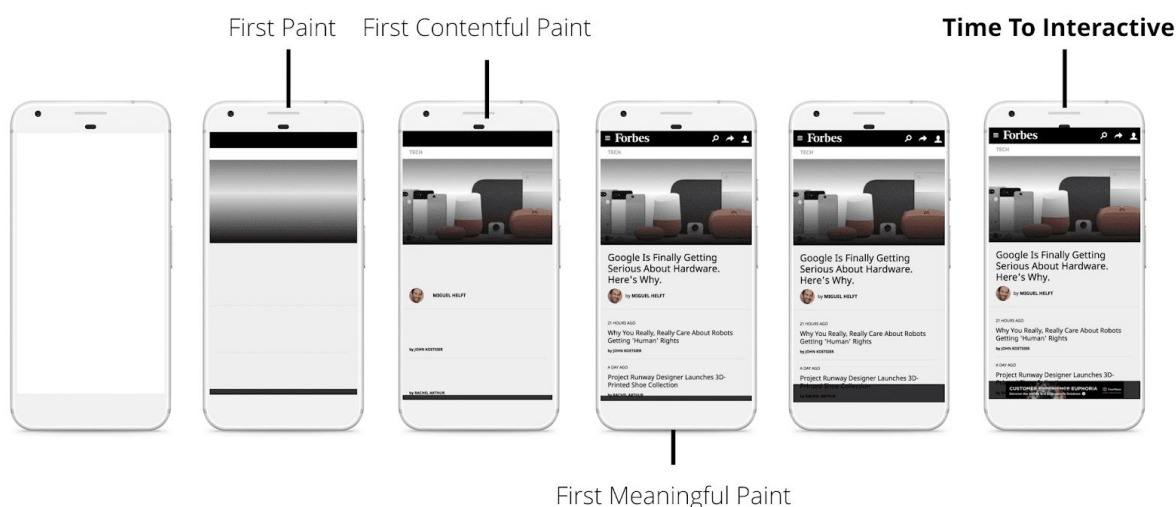
1. Use el tiempo de inactividad para completar el trabajo diferido. Por ejemplo, para la carga de la página inicial, cargue la menor cantidad de datos posible, luego use el tiempo de inactividad para cargar el resto.
2. Realice el trabajo durante el tiempo de inactividad en 50ms o menos. Por más tiempo, corre el riesgo de interferir con la capacidad de la aplicación de responder a la entrada del usuario en 50 ms.
3. Si un usuario interactúa con una página durante el trabajo de inactividad, la interacción del usuario siempre debe tener la máxima prioridad e interrumpir el trabajo de tiempo inactivo.

● Carga: entrega contenido interactivo en menos de 5 segundos

Cuando las páginas se cargan lentamente, la atención del usuario vaga, y los usuarios perciben que la tarea está rota. Los sitios que se cargan rápidamente tienen sesiones promedio más largas, tasas de rebote más bajas y una mayor visibilidad de anuncios. Vea La necesidad de la velocidad móvil: cómo la latencia móvil afecta los ingresos de los editores.

Metas:

- Optimice para un rendimiento de carga rápido en relación con el dispositivo y las capacidades de red que utilizan los usuarios para acceder a su sitio. Actualmente, un buen objetivo para las primeras cargas es cargar la página y ser interactivo en 5 segundos o menos en dispositivos móviles de rango medio con conexiones 3G lentas. Vea ¿Puede permitirse? Presupuestos de rendimiento web en el mundo real. Pero tenga en cuenta que estos objetivos pueden cambiar con el tiempo.
- Para cargas posteriores, un buen objetivo es cargar la página en menos de 2 segundos. Pero este objetivo también puede cambiar con el tiempo.



Pautas:

1. Pon a prueba el rendimiento de tu carga en los dispositivos móviles y las conexiones de red que son comunes entre tus usuarios. Si su empresa tiene información sobre en qué dispositivos y conexiones de red están sus usuarios, puede usar esa combinación y establecer sus propios objetivos de rendimiento de carga. De lo contrario, The Mobile Economy 2017 sugiere que una buena base de referencia global es un teléfono Android de rango medio, como un Moto G4 y una red 3G lenta, definida como 400 ms de RTT y velocidad de transferencia de 400 kbps. Esta combinación está disponible en WebPageTest.
2. Tenga en cuenta que aunque el dispositivo de su usuario móvil típico podría afirmar que está en una conexión 2G, 3G o 4G, en realidad la velocidad de conexión efectiva es a menudo significativamente más lenta, debido a la pérdida de paquetes y la varianza de la red.

3. Concéntrese en optimizar la Ruta de reproducción crítica para desbloquear la representación.
4. No tiene que cargar todo en menos de 5 segundos para producir la percepción de una carga completa. Habilite la representación progresiva y trabaje en segundo plano. Deferir cargas no esenciales a períodos de tiempo de inactividad. Consulte Optimización del rendimiento del sitio web.
5. Reconozca los factores que afectan el rendimiento de carga de la página:
 - a. Velocidad de red y latencia
 - b. Hardware (CPU más lentas, por ejemplo)
 - c. Desalojo de caché
 - d. Diferencias en el almacenamiento en caché L2 / L3
 - e. Análisis de JavaScript

ANALIZANDO PERFORMANCE

Para comenzar nuestro debugging vamos a abrir Google Chrome en modo de incógnito. El modo incógnito garantiza que Chrome se ejecute en un estado limpio. Por ejemplo, si tiene muchas extensiones instaladas, esas extensiones pueden crear ruido en sus mediciones de rendimiento.

Alternativamente se puede probar cargando la siguiente página, la cual ya está pre programada para poder ser analizada :

`https://googlechrome.github.io/devtools-samples/jank/`

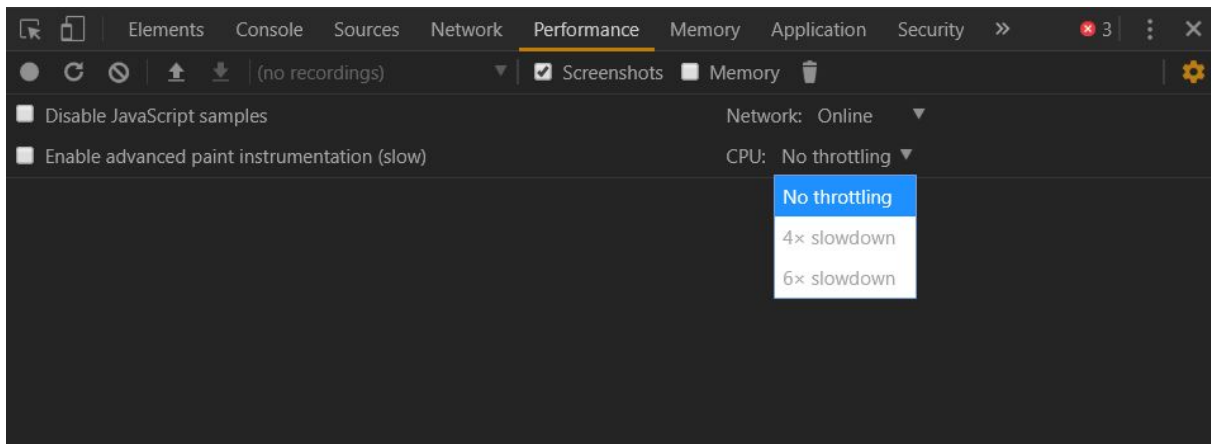
Dentro de la consola de desarrollo del navegador vamos a la pestaña Performance para poder comenzar a analizar nuestro programa.

Simular una CPU móvil

Los dispositivos móviles tienen mucha menos potencia de CPU que los equipos de escritorio y las computadoras portátiles. Siempre que diseñe una página, use la aceleración de CPU para simular cómo se desempeña su página en los dispositivos móviles.

- En DevTools, haz clic en la pestaña Rendimiento.
- Asegúrese de que la casilla de verificación Capturas de pantalla esté habilitada.


- Haga clic en Ajustes de captura Configuración de captura. DevTools revela configuraciones relacionadas con la forma en que captura las métricas de rendimiento.
- Para CPU, seleccione 2x ralentización. DevTools acelera tu CPU para que sea 2 veces más lenta de lo normal.




Grabar el rendimiento del tiempo de ejecución

Cuando ejecutó la versión optimizada de la página, los cuadrados azules se mueven más rápido. ¿Porqué es eso? Se supone que ambas versiones mueven cada cuadrado la misma cantidad de espacio en la misma cantidad de tiempo. Haga una grabación en el panel de rendimiento para aprender a detectar el cuello de botella de rendimiento en la versión no optimizada.

- En DevTools, haga clic en Grabar registro. DevTools captura las métricas de rendimiento a medida que se ejecuta la página.

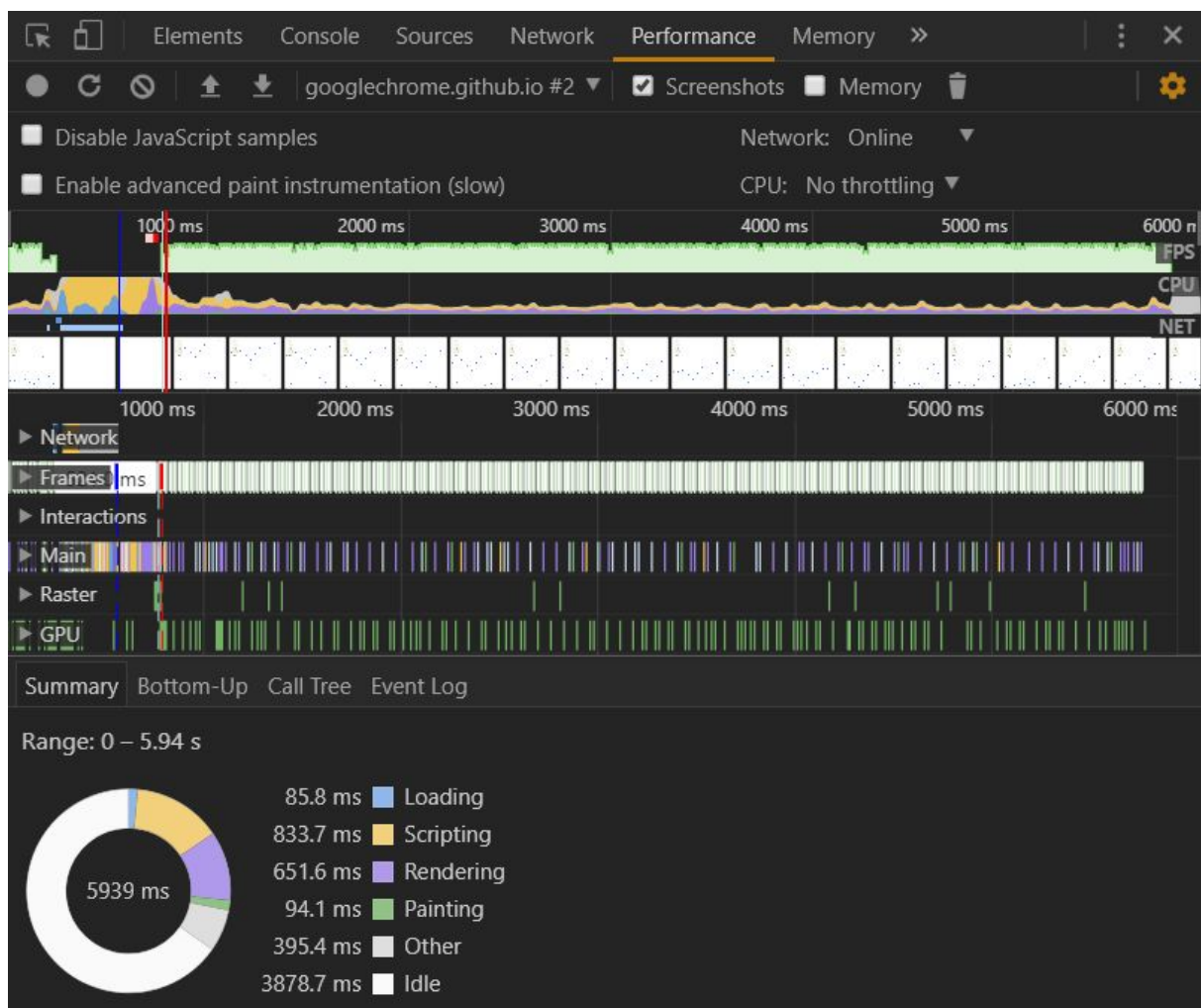
Click the record button  or hit **Ctrl + E** to start a new recording.

Click the reload button  or hit **Ctrl + Shift + E** to record the page load.

After recording, select an area of interest in the overview by dragging.
Then, zoom and pan the timeline with the mousewheel or **WASD** keys.

[Learn more](#)

- Espere unos segundos.
- Haga clic en Detener. DevTools detiene la grabación, procesa los datos y luego muestra los resultados en el panel Rendimiento.
-



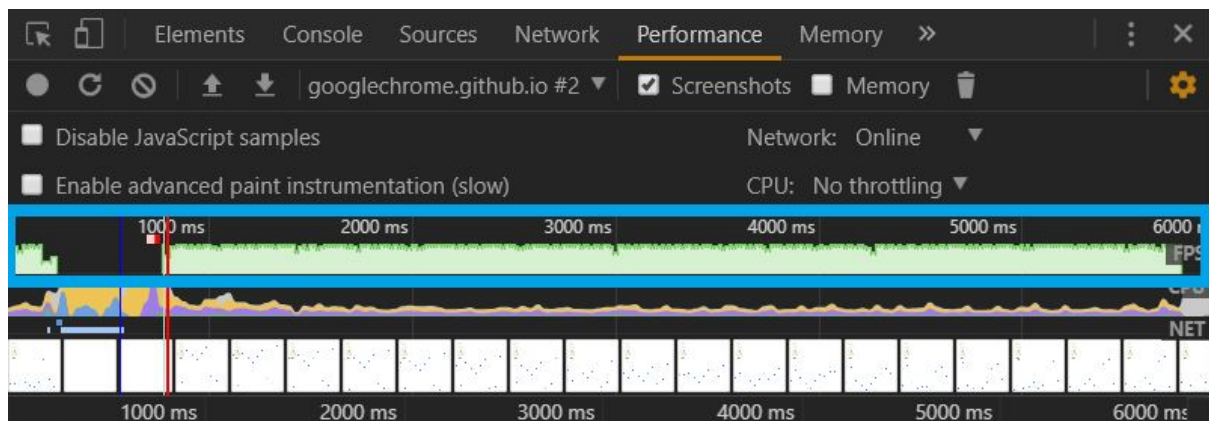
Analiza los resultados

Una vez que tenga una grabación del rendimiento de la página, puede medir qué tan pobre es el rendimiento de la página y encontrar la (s) causa (s) ⁷.

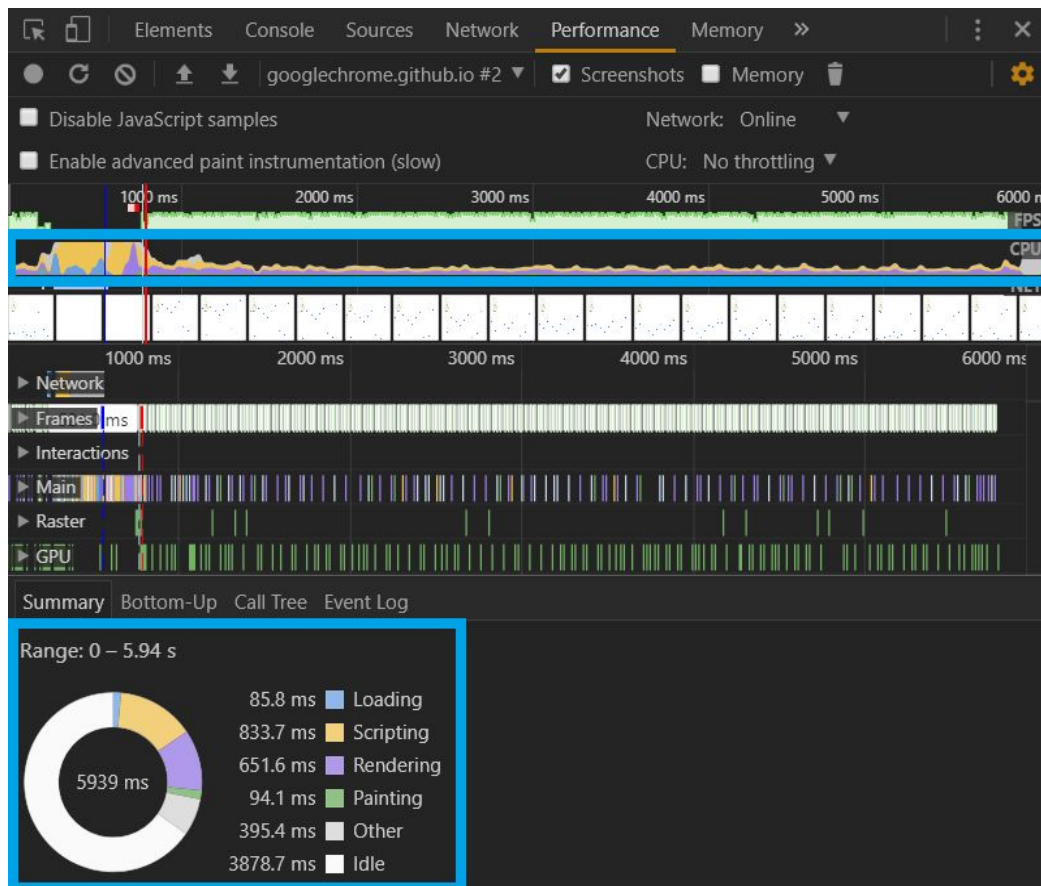
Analizar fotogramas por segundo

La métrica principal para medir el rendimiento de cualquier animación es fotogramas por segundo (FPS). Los usuarios están felices cuando las animaciones se ejecutan a 60 FPS.

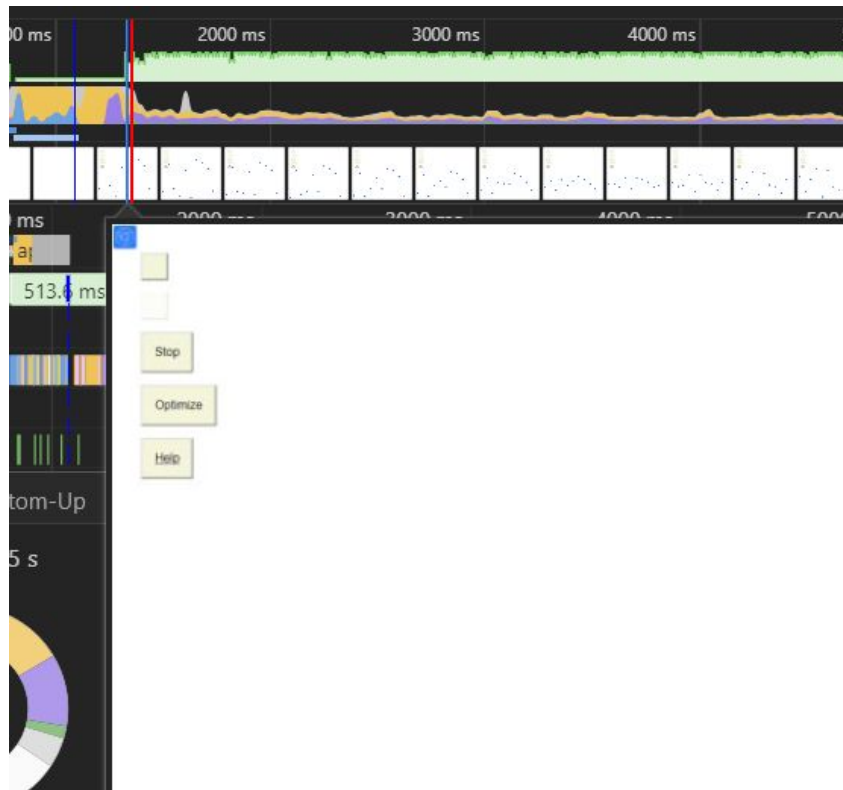
- Mire la tabla de FPS. Cada vez que ve una barra roja por encima de FPS, significa que la velocidad de fotogramas bajó tanto que probablemente esté dañando la experiencia del usuario. En general, cuanto mayor es la barra verde, mayor es el FPS.



- Debajo del gráfico de FPS, verá la tabla de CPU. Los colores en el gráfico de CPU corresponden a los colores en la pestaña Resumen, en la parte inferior del panel Rendimiento. El hecho de que el diagrama de la CPU esté lleno de color significa que la CPU se agotó al máximo durante la grabación. Cada vez que vea que la CPU está al máximo durante largos períodos, es una señal para encontrar formas de hacer menos trabajo.



- Pase el mouse sobre los gráficos FPS, CPU o NET. DevTools muestra una captura de pantalla de la página en ese momento. Mueva su mouse hacia la izquierda y hacia la derecha para reproducir la grabación. Esto se conoce como depuración, y es útil para analizar manualmente la progresión de las animaciones.

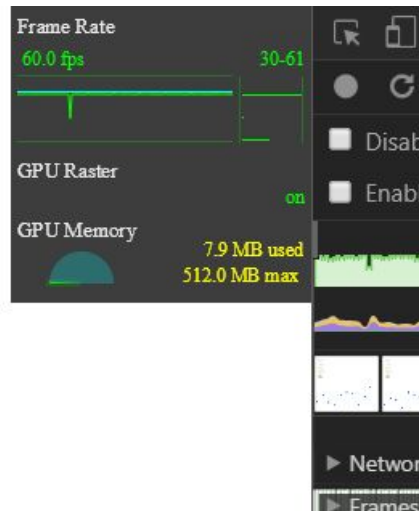


- En la sección Frames, coloque el mouse sobre uno de los cuadrados verdes. DevTools te muestra el FPS para ese marco en particular. Cada cuadro está probablemente muy por debajo del objetivo de 60 FPS.

Bonus: abre el medidor FPS

Otra herramienta útil es el medidor FPS, que proporciona estimaciones en tiempo real para FPS a medida que se ejecuta la página.

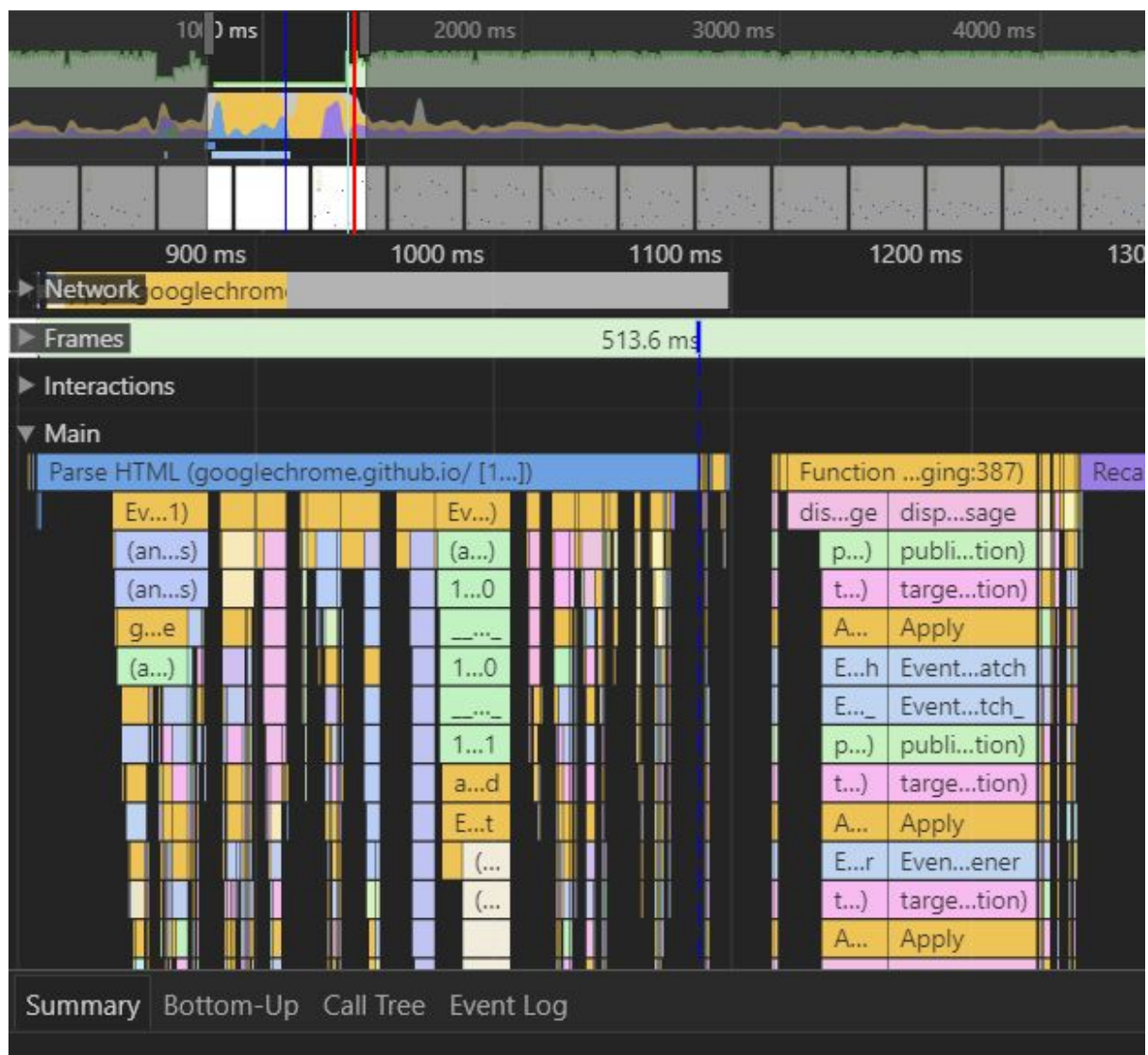
- Presione Comando + Shift + P (Mac) o Control + Shift + P (Windows, Linux) para abrir el Menú de Comando.
- Comience a escribir Rendering en el menú de comandos y seleccione Mostrar representación.
- En la pestaña Rendering, habilite FPS Meter. Aparece una nueva superposición en la parte superior derecha de su ventana gráfica.



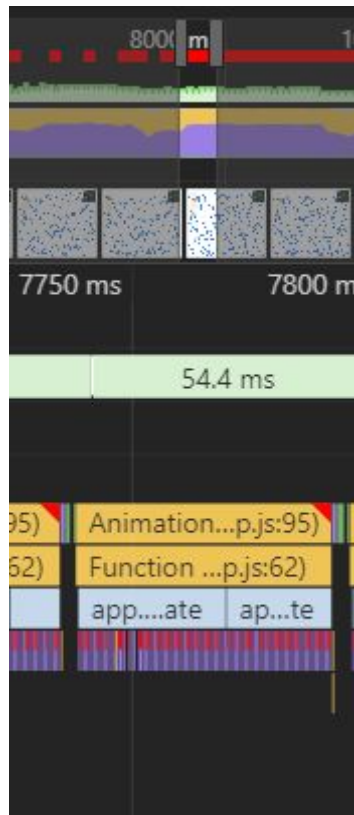
Encuentra el cuello de botella

Ahora que ha medido y verificado que la animación no está funcionando bien, la siguiente pregunta para responder es: ¿por qué?

1. Tenga en cuenta la pestaña de resumen. Cuando no se seleccionan eventos, esta pestaña muestra un desglose de la actividad. La página pasó la mayor parte de su tiempo de representación. Como el rendimiento es el arte de hacer menos trabajo, su objetivo es reducir la cantidad de tiempo invertido en el trabajo de renderizado.
2. Expande la sección Main. DevTools le muestra una tabla de llama de actividad en el hilo principal, a lo largo del tiempo. El eje x representa la grabación, a lo largo del tiempo. Cada barra representa un evento. Una barra más ancha significa que el evento tomó más tiempo. El eje y representa la pila de llamadas. Cuando ve eventos apilados uno encima del otro, significa que los eventos superiores causaron los eventos más bajos.



- Hay una gran cantidad de datos en la grabación. Amplíe un solo evento de Marco de animación disparado haciendo clic, manteniendo presionado y arrastrando el mouse sobre la Descripción general, que es la sección que incluye los gráficos FPS, CPU y NET. La sección Principal y la pestaña Resumen solo muestran información para la parte seleccionada de la grabación.
- Tenga en cuenta el triángulo rojo en la parte superior derecha del evento de Marco de animación disparado. Cada vez que vea un triángulo rojo, es una advertencia de que puede haber un problema relacionado con este evento.



REFLOW - REPAINT

Otra consideración del rendimiento, que suele ser el factor más importante para muchas actualizaciones de estilo, es el gran volumen de trabajo que debe llevarse a cabo cuando cambia un elemento.

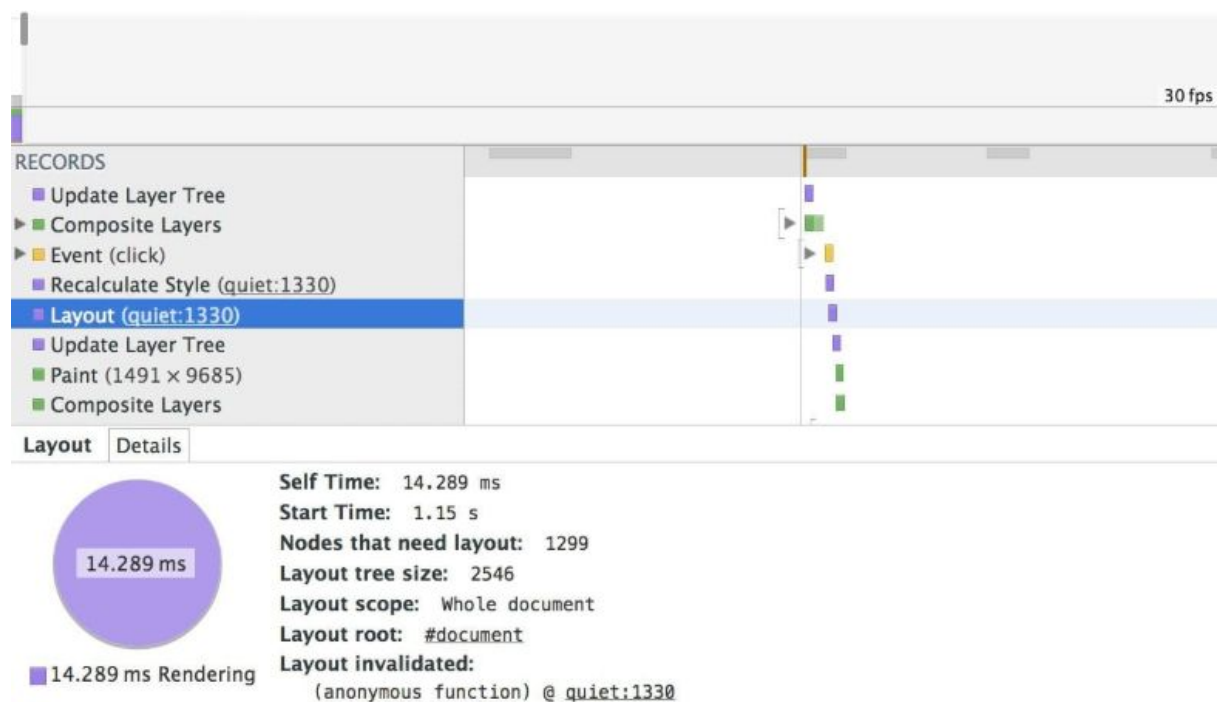
En términos generales, el peor de los casos para calcular el estilo calculado de los elementos es la cantidad de elementos multiplicados por el recuento de selectores, ya que cada elemento debe verificarse al menos una vez con cada estilo para ver si coincide.

Los cálculos de estilo a menudo se pueden dirigir a unos pocos elementos directamente en lugar de invalidar la página como un todo. En los navegadores modernos esto tiende a ser un problema mucho menos importante, ya que el navegador no necesariamente necesita verificar todos los elementos potencialmente afectados por un cambio. Los navegadores antiguos, por otro lado, no están necesariamente tan optimizados para tales tareas. Donde pueda, debe reducir el número de elementos invalidados.

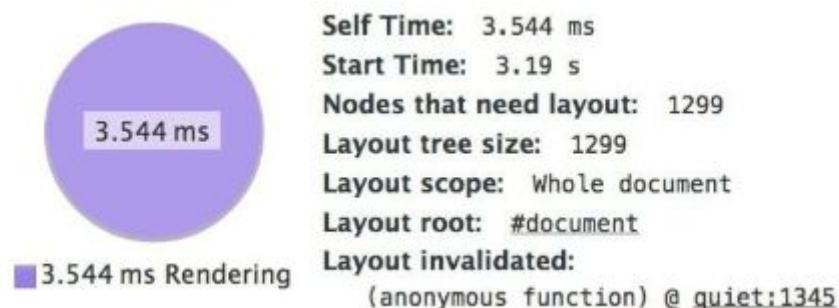
Usa flexbox sobre modelos de diseño más antiguos

La web tiene una gama de modelos de diseño, algunos son más compatibles que otros. El modelo de diseño de CSS más antiguo nos permite posicionar elementos en la pantalla de forma relativamente absoluta y mediante elementos flotantes.

La siguiente captura de pantalla muestra el costo de diseño al usar flotadores en 1.300 cajas. Es, sin dudas, un ejemplo artificial, porque la mayoría de las aplicaciones usarán una variedad de medios para colocar elementos.

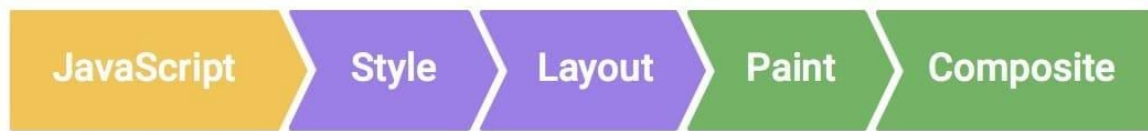


Si actualizamos la muestra para usar Flexbox, una adición más reciente a la plataforma web, obtenemos una imagen diferente:



Evite diseños síncronos forzados

El envío de un marco a la pantalla tiene este orden:



Primero se ejecuta el JavaScript, luego los cálculos de estilo, luego el diseño. Sin embargo, es posible obligar a un navegador a realizar un diseño anterior con JavaScript. Se llama un diseño síncrono forzado.

Lo primero a tener en cuenta es que, a medida que JavaScript se ejecuta, todos los valores de diseño antiguos del marco anterior son conocidos y están disponibles para su consulta. Entonces, si, por ejemplo, desea escribir la altura de un elemento (llamémoslo "caja") al comienzo del cuadro, puede escribir un código como este:

```
console.log(box.offsetHeight);
```

Las cosas se vuelven problemáticas si ha cambiado los estilos de la caja antes de pedir su altura:

```
box.classList.add('super-big');  
console.log(box.offsetHeight);
```

Ahora, para responder la pregunta de altura, el navegador primero debe aplicar el cambio de estilo (debido a que agrega la clase súper grande) y luego ejecutar el diseño. Solo entonces podrá regresar la altura correcta. Este es un trabajo innecesario y potencialmente costoso.

Debido a esto, siempre debe lotear las lecturas de estilo y hacerlas primero (donde el navegador puede usar los valores de diseño del marco anterior) y luego hacer cualquier escritura.

En su mayor parte, no debería necesitar aplicar estilos y luego consultar valores; usar los valores del último cuadro debe ser suficiente. Ejecutar los cálculos de estilo y el diseño de forma sincrónica y antes de lo que el navegador quisiera son posibles cuellos de botella, y no es algo que normalmente querría hacer.

Evite la manipulación de diseño

Hay una forma de hacer que los diseños síncronos forzados sean aún peores: haz muchos de ellos en rápida sucesión. Eche un vistazo a este código:

```
function resizeAllParagraphsToMatchBlockWidth() {
  for (var i = 0; i < paragraphs.length; i++) {
    paragraphs[i].style.width = box.offsetWidth + 'px';
  }
}
```

Este código pasa por un grupo de párrafos y establece el ancho de cada párrafo para que coincida con el ancho de un elemento llamado "cuadro". Parece inofensivo, pero el problema es que cada iteración del ciclo lee un valor de estilo (box.offsetWidth) y luego lo usa de inmediato para actualizar el ancho de un párrafo (párrafos [i].style.width). En la siguiente iteración del ciclo, el navegador debe tener en cuenta el hecho de que los estilos han cambiado desde que se solicitó por última vez offsetWidth (en la iteración anterior), por lo que debe aplicar los cambios de estilo y ejecutar el diseño. Esto sucederá en cada iteración!.

La solución para esta muestra es leer nuevamente y luego escribir valores:

```
//Leer
var width = box.offsetWidth;

function resizeAllParagraphsToMatchBlockWidth() {
  for (var i = 0; i < paragraphs.length; i++) {
    // Ahora Escribir.
    paragraphs[i].style.width = width + 'px';
  }
}
```

1. <https://developer.mozilla.org/en-US/docs/Web/API/Storage>
2. <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>
3. <https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>
4. <https://developer.mozilla.org/en-US/docs/Web/API/Storage/setItem>
5. <https://developer.mozilla.org/en-US/docs/Web/API/Storage/getItem>
6. <https://developers.google.com/web/fundamentals/performance/rail#devtools>
7. https://developers.google.com/web/tools/chrome-devtools/rendering-tools/#paint_and_composite