Centro de Tecnología y Artes Visuales

Diseño de Interfaces de Usuario

Profesor
Andrés Díaz Soto

Autor
Saúl López Molina

2019

Lazy loading images

El navegador utiliza el atributo **src** de la etiqueta para activar la carga de la imagen..
Si el navegador obtiene el atributo **src**, activará la descarga de la imagen,
Para diferir la carga, se coloca la URL de la imagen en un atributo que no sea **src**.
por ejemplo en el atributo **data-src** de la etiqueta de la imagen. Ahora que **src** está vacío y el navegador no activará la carga de la imagen luego hay que decirle al navegador cuándo debe cargarla. Para esto, se verifica que tan pronto como la imagen ingresa a la ventana gráfica, se activa la carga.

Hay dos formas de verificar cuándo una imagen entra en la ventana gráfica.

Método 1: desencadenar la carga de imágenes usando eventos de Javascript
Esta técnica utiliza event listener en el **scroll**, **resize** y **orientationChange** en el navegador
Cuando ocurre alguno de estos eventos, se encuentran todas las imágenes en la página que se han diferido y, a partir de estas imágenes, se verifica cuáles están actualmente en la ventana gráfica. Esto se hace usando el desplazamiento superior de la imagen, la posición superior del documento actual y la altura de la ventana. Si una imagen ha entrado en la ventana gráfica, se selecciona la URL del atributo **data-src** y la movemos al atributo **src** y la imagen se cargará como resultado.

Tenga en cuenta que se le pide a JavaScript que seleccione imágenes que contengan una clase **lazy**. Una vez que la imagen se haya cargado, se elimina la clase porque ya no necesita activar un evento. Y, una vez que se cargan todas las imágenes, también se eliminan los event Listeners.

Método 2: desencadenar la carga de imágenes usando la API de Intersection Observer

Este método facilita la detección cuando un elemento entra en la ventana gráfica y realiza una acción cuando lo hace. En este método se adjunta el observer en todas las imágenes para que se carguen de forma perezosa. Una vez que la API detecta que el elemento ha entrado en la ventana gráfica, utilizando la propiedad **isIntersecting** se selecciona la URL de atributo **data-src** y se mueve al atributo **src** atributo para que el navegador active la carga de la imagen. Una vez hecho esto, se elimina la clase **lazy** y el **observer** de la imagen

Intersection Observer API

La API Intersection observer le permite configurar un callback que se llama cuando un elemento, llamado **target,** se interseca con el viewport o con un elemento específico, para los fines de esta API, esto llama al **root element** o **root** Por lo general, deseará observar los cambios de intersección con respecto al antepasado desplazable más cercano del elemento o, si el elemento no es descendiente de un elemento desplazable, el viewport. Para observar la intersección relativa al root element, específicamente NULL.
El grado de intersección entre el target element y su root es la **intersection ratio**. Esta es una representación del porcentaje del target element que es visible como un valor entre 0.0 y 1.0.

Creando un Intersection observer

Cree el observador de intersección llamando a su constructor y pasándole una función callback para que se ejecute siempre que se cruce en una dirección o en otra:

```
var options = {
  root: document.querySelector('#scrollArea'),
  rootMargin: '0px',
  threshold: 1.0
}

var observer = new IntersectionObserver(callback, options);
```

Un umbral de 1.0 significa que cuando el 100% del objetivo es visible dentro del elemento especificado por la opción **root**, se invoca el callback.
Una vez que haya creado el observador, debe asignarle un elemento de destino para ver:

```
var target = document.querySelector('#listItem');
observer.observe(target);
```

Cuando el destino cumple un límite especificado para **IntersectionObserver**, se invoca el callback que recibe una lista de objetos **IntersectionObserverEntry** y el observador:

```
var callback = function(entries, observer) {
  entries.forEach(entry => {
    // Each entry describes an intersection change for one observed
    // target element:
    //   entry.boundingClientRect
    //   entry.intersectionRatio
    //   entry.intersectionRect
    //   entry.isIntersecting
    //   entry.rootBounds
    //   entry.target
    //   entry.time
  });
};
```