Programación

Temporalizadores

Vigésima semana

Marzo 2022

Cruz García, Iago

Introducción

La temporalización de acciones en las aplicaciones web permite establecer rutinas temporales, añadir bloqueos o activadores temporales… en general, modificar el comportamiento de nuestros programas en un alto grado.

Recomendación: A partir de ahora es necesario que a la hora de realizar ejercicios nos acostumbremos a buscar información en la documentación oficial. En caso de no poder resolver las dudas con la documentación ofrecida, el siguiente paso será preguntar las dudas en el foro de clase.

Documentación oficial de JavaScript

Foro de la asignatura

Documentación acerca de los intervalos

Documentación acerca de los temporizadores

Control de tiempo

Temporizadores

Los temporizadores nos permiten indicar el tiempo de retardo para ejecutar un programa. Para utilizarlos usaremos las funciones TimeOut:

- **setTimeout(*función, retardo, parámetros):*** Establece que la **función** indicada se lance tras un **retardo** enviando los **parámetros** indicados. El **retardo** se mide en milisegundos.
- **clearTimeout(Timeout)**: Elimina el temporizador **Timeout** indicado.

Una cosa a tener en cuenta con estos métodos, es la necesidad de almacenar la función "setTimeout" en una variable para su posible manejo posterior. A continuación se muestra un ejemplo de un alert que se ejecuta tras dos segundos de cargar la página.

main.js

```
function aviso(str) {
	alert(str);
}

function __main__() {
	var temporizador = setTimeout(aviso, 2000, "Esto es un temporizador");
	//clearTimeout(temporizador);
}
__main__();
```

La función 'clearTimeout(temporizador)' está comentada para evitar que se borre el aviso, pero si probais descomentarla, vereis que dicha función nunca se ejecuta.

Este tipo de funciones suelen utilizarse para bloquear botones o campos de texto durante un tiempo determinado tras realizar una acción. Para bloquear un botón tras pulsarlo durante dos segundos, tenéis un ejemplo:

main.js

```
function habilitar(target) {
	console.log("fun")
	target.disabled = false;
}

function __main__() {
	var boton = document.getElementById("boton");
	boton.addEventListener("click", (evt) => {
		evt.currentTarget.disabled = true;
		var temp = setTimeout(habilitar, 2000, evt.currentTarget);
	});
}
```

Intervalos

La creación de intervalos permite ejecutar un programa o bloque de código cada cierto intervalo mientras esté activo. Utilizaremos funciones de **Interval** para hacer esto:

- **setInterval(*función, intervalo, parámetros)*:** Establece que la **función** especificada se ejecute cada **intervalo** de tiempo con los **parámetros** designados. El **intervalo** es medido en milisegundos.
- **clearInterval(Interval):** Elimina el intervalo **Interval** en ejecución.

Igual que con **Timeout**, para poder utilizar 'clearInterval' necesitaremos almacenar la función 'setInterval' en una variable. A continuación se muestra un ejemplo de una función que actualiza un reloj cada segundo.

main.js

```
function reloj() {
        var par = document.getElementById("parrafo");
        var fecha = new Date();
        var hora = fecha.getHours();
        var minutos = fecha.getMinutes();
        var segundos = fecha.getSeconds();
        par.innerHTML = hora + ":" + minutos + ":" + segundos;
}

function __main__() {
        var intervalo = setInterval(reloj, 1000);
}
```

Una estructura de los intervalos muy interesante es la siguiente:

main.js

```
function activarReloj() {
        var par = document.getElementById("parrafo");
        var fecha = new Date();
        var hora = fecha.getHours();
        var minutos = fecha.getMinutes();
        var segundos = fecha.getSeconds();
        par.innerHTML = hora + ":" + minutos + ":" + segundos;
}

function reloj(intervalo) {
        var inter = intervalo;
        var relojActivo = parseInt(localStorage.getItem("relojActivo"));
        relojActivo = Number.isNaN(relojActivo) ? 0 : relojActivo;
        if (relojActivo == 0) {
          relojActivo = 1;
          localStorage.setItem("relojActivo", relojActivo);
          activarReloj();
          inter = setInterval(activarReloj, 1000);
        } else {
          relojActivo = 0;
          localStorage.setItem("relojActivo", relojActivo);
          clearInterval(inter);
        }
        return inter;
}

function __main__() {
        var intervalo;
        var boton = document.getElementById("boton");
        boton.addEventListener("click", () => {
           intervalo = reloj(intervalo);
        });
        activarReloj();
}
```

```
__main__();
```

La función **reloj** es la encargada de comprobar si el botón para parar o continuar el reloj se pulsó. Si el reloj está en funcionamiento (por lo tanto, el valor es 1 de relojActivo), se para. Si está sin funcionar (relojActivo igual a 0), se activará. El valor de relojActivo podría ser una variable global, pero se utiliza localStorage para mantener la ejecución entre sesiones.

Conclusión y consejos

Como veis, es relativamente sencillo poner a funcionar un temporizador o un intervalo. El mayor de los problemas es controlar su ejecución a posteriori. A continuación os dejo una serie de consejos a tener en cuenta cuando se trabajen con estas funciones:

- Siempre debería guardarse la función de **setTimeout y setInterval** en una variable para poder parar su ejecución si fuera necesario.
- Al igual que los eventos, no es buena idea establecer temporizadores o intervalos anidados, unos dentro de otros, pues se tiende a perder el control de la ejecución rápidamente.
- Utilizar variables para controlar el flujo de los eventos es imprescindible, pero se pueden utilizar otros métodos como, por ejemplo, un botón para iniciar y otro para parar el intervalo. Siempre hay que adaptarlo al diseño y uso del programa.

- Los intervalos como tal no se pueden almacenar en memoria semi-volátil o no-volátil en JavaScript, a pesar de poder almacenarse en una variable volátil. Tened cuidado con esto para evitar almacenar datos incoherentes.