EBook Gratis

APRENDIZAJE web-audio

Free unaffiliated eBook created from
Stack Overflow contributors.

#web-audio

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: web-audio

It is an unofficial and free web-audio ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official web-audio.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con web-audio

Observaciones

La Web Audio API es un estándar W3C para el acceso de nivel inferior al sistema de audio que la etiqueta estándar `<audio>` , a través de una API de alto nivel.

Los casos de uso incluyen juegos, arte, síntesis de audio, aplicaciones interactivas, producción de audio y cualquier aplicación donde se requiera un control preciso de los datos de audio.

La API puede aceptar entradas de varias fuentes, incluida la carga de archivos de audio y la decodificación a través de la API o `<audio>` -elementos. También proporciona facilidades para generar sonido directamente a través de la API mediante el uso de nodos de oscilador.

También hay una serie de nodos de procesamiento, como ganancias, retrasos y procesadores de scripts (que eventualmente serán desaprobados y reemplazados por nodos más eficientes). Estos a su vez pueden usarse para construir efectos más complejos y gráficos de audio.

Examples

Sintetizando audio

En este ejemplo, mostramos cómo generar una onda sinusoidal simple y emitirla en los parlantes / auriculares del usuario.

```
let audioContext = new (window.AudioContext || window.webkitAudioContext)();

let sourceNode = audioContext.createOscillator();
sourceNode.type = 'sine';
sourceNode.frequency.value = 261.6;
sourceNode.detune.value = 0;

//Connect the source to the speakers
sourceNode.connect(audioContext.destination);

//Make the sound audible for 100 ms
sourceNode.start();
window.setTimeout(function() { sourceNode.stop(); }, 100);
```

Los métodos de `start` y `stop` de la variable `sourceNode` encima de cada uno tienen un parámetro opcional `when` eso especifica cuántos **segundos se** deben esperar antes de comenzar o detenerse.

Entonces, una forma alternativa de detener el sonido sería:

```
sourceNode.start();
sourceNode.stop(0.1);
```

El parámetro de `type` de un nodo de oscilador se puede establecer en cualquiera de los siguientes valores:

- seno (por defecto)
- cuadrado
- diente de sierra
- traingle
- una ola personalizada

Las ondas personalizadas son `PeriodicWaves` y se pueden crear utilizando el método `AudioContext.createPeriodicWave` .

Usando efectos en audio

Los efectos se pueden aplicar al audio encadenando nodos entre el origen y el nodo de destino. En este ejemplo, usamos un nodo de ganancia para silenciar la fuente, y solo dejamos pasar el sonido en momentos específicos. Esto nos permite crear código morse.

```
function morse(gainNode, pattern) {
  let silenceTimeout = 300;
  let noiseTimeout;

  if(pattern === '') {
    //We are done here
    return;
  } else if(pattern.charAt(0) === '.') {
    noiseTimeout = 100;
  } else if(pattern.charAt(0) === '-') {
    noiseTimeout = 400;
  } else {
    console.error(pattern.charAt(0), ': Character not recognized.');
    return;
  }

  //Briefly let sound through this gain node
  gainNode.gain.value = 1;
  window.setTimeout(function() {
    gainNode.gain.value = 0;
    window.setTimeout(morse, silenceTimeout, gainNode, pattern.substring(1));
  }, noiseTimeout);
}
```

```
let audioContext = new (window.AudioContext || window.webkitAudioContext)();

let sourceNode = audioContext.createOscillator();
let gainNode = audioContext.createGain();

sourceNode.type = 'sine';
sourceNode.frequency.value = 261.6;
sourceNode.detune.value = 0;

//Mute sound going through this gain node
gainNode.gain.value = 0;

//SourceNode -> GainNode -> Speakers
```

```
sourceNode.connect(gainNode);
gainNode.connect(audioContext.destination);

//The source node starts outputting
sourceNode.start();

//Output SOS
morse(gainNode, '...---...');
```

Grabación de audio desde una fuente de micrófono

Para grabar audio desde el micrófono de un usuario, primero debemos obtener el permiso del usuario para acceder al dispositivo:

```
navigator.mediaDevices.getUserMedia({ audio: true })
    .then(successCallback)
    .catch(failureCallback);
```

En caso de éxito, nuestra `successCallback` se llamará con un `MediaStream` objeto que podemos utilizar para acceder al micrófono:

```
var audioContext = new (window.AudioContext || window.webkitAudioContext)();
// Create a source from our MediaStream
var source = audioContext.createMediaStreamSource(mediaStream);
// Now create a Javascript processing node with the following parameters:
// 4096 = bufferSize (See notes below)
// 2 = numberOfInputChannels (i.e. Stereo input)
// 2 = numberOfOutputChannels (i.e. Stereo output)
var node = audioContext.createScriptProcessor(4096, 2, 2);
node.onaudioprocess = function(data) {
    console.log(data);
}
// Connect the microphone to the script processor
source.connect(node);
node.connect(audioContext.destination);
```

El evento `onaudioprocess` nos da acceso al flujo de datos Raw PCM desde el micrófono. Podemos acceder a los datos almacenados de esta manera:

```
node.onaudioprocess = function(data) {
    var leftChannel = data.inputBuffer.getChannelData(0).buffer;
    var rightChannel = data.inputBuffer.getChannelData(1).buffer;
}
```

Cuando hayamos terminado de grabar, debemos desconectarnos de la fuente y descartar el procesador de scripts:

```
node.disconnect();
source.disconnect();
```

Notas importantes sobre `bufferSize`

El `bufferSize` determina con qué frecuencia se `onaudioprocess` devolución de llamada

`onaudioprocess` . Los valores más bajos producen una latencia más baja (mejor), los valores más altos reducirán las interrupciones / fallos de audio. Un valor de cero permitirá que la implementación del navegador elija un valor apropiado. Si se pasa manualmente, debe ser uno de los siguientes valores: 256, 512, 1024, 2048, 4096, 8192, 16384.

Reproducción de audio

Para reproducir audio utilizando la Web Audio API, necesitamos obtener un ArrayBuffer de datos de audio y pasarlo a un BufferSource para su reproducción.

Para obtener un búfer de audio del sonido a reproducir, debe usar el método `AudioContext.decodeAudioData` manera:

```
const audioCtx = new (window.AudioContext || window.webkitAudioContext)();
// Fetch the MP3 file from the server
fetch("sound/track.mp3")
    // Return the data as an ArrayBuffer
    .then(response => response.arrayBuffer())
    // Decode the audio data
    .then(buffer => audioCtx.decodeAudioData(buffer))
    .then(decodedData => {
        // ...
    });
```

Cuando se resuelva la promesa final, se le entregará el audio en forma de un `AudioBuffer` . Esto se puede adjuntar a un `AudioBufferSourceNode` , y reproducirse, por ejemplo:

```
const source = context.createBufferSource();
source.buffer = decodedData; // <==
source.start(...);
```

Donde `.start()` tiene tres parámetros que compensan cuándo reproducir la muestra, compensan desde dónde jugar en la muestra y dicen cuánto tiempo se debe reproducir la muestra, respectivamente.

Puede encontrar más información sobre cómo manipular el origen del búfer en MDN .

Alteración en tiempo real de dos fuentes de audio.

Este ejemplo muestra cómo utilizar dos fuentes de audio y modificar una de ellas según la otra. En este caso, creamos un audio Ducker, que bajará el volumen de la pista principal si la pista secundaria produce sonido.

El ScriptProcessorNode enviará eventos regulares a su controlador de audioprocesos. En este controlador, que está vinculado a la fuente de audio secundaria, calculamos la "sonoridad" del audio, y lo usamos para alterar un compresor dinámico en la fuente de audio primaria. Ambos se envían a los parlantes / auriculares del usuario. El resultado son cambios de volumen muy abruptos en la pista de audio primaria cuando se detecta sonido en la pista de audio secundaria. Podríamos hacer esto más suave usando un promedio y usando una línea de retardo para cambiar el volumen antes de que se detecte el audio secundario, pero el proceso debería ser

claro en este ejemplo.

```
//The DuckingNode will heavily compress the primary source when the secondary source produces
sound
class DuckingNode {
  constructor(context) {
    let blocksize = 2048;
    let normalThreshold = -50;
    let duckThreshold = 0.15;

    //Creating nodes
    this.compressor = context.createDynamicsCompressor();
    this.processor = context.createScriptProcessor(blocksize, 2, 2);

    //Configuring nodes
    this.compressor.threshold.value = normalThreshold;
    this.compressor.knee.value = 0;
    this.compressor.ratio.value = 12;
    this.compressor.reduction.value = -20;
    this.compressor.attack.value = 0;
    this.compressor.release.value = 1.5;

    let self = this;
    this.processor.onaudioprocess = function(audioProcessingEvent) {
      let inputBuffer = audioProcessingEvent.inputBuffer;
      let outputBuffer = audioProcessingEvent.outputBuffer;
      let rms;
      let total = 0.0;
      let len = blocksize * outputBuffer.numberOfChannels;

      for (let channel = 0; channel < outputBuffer.numberOfChannels; channel++) {
        let inputData = inputBuffer.getChannelData(channel);
        let outputData = outputBuffer.getChannelData(channel);

        for (let sample = 0; sample < inputBuffer.length; sample++) {
          // make output equal to the same as the input
          outputData[sample] = inputData[sample];
          total += Math.abs(inputData[sample]);
        }
      }

      //Root mean square
      rms = Math.sqrt( total / len );

      //We set the threshold to at least 'normalThreshold'
      self.compressor.threshold.value = normalThreshold + Math.min(rms - duckThreshold, 0) * 5
* normalThreshold;
    }
  }

  get primaryInput () {
    return this.compressor;
  }

  get secondaryInput () {
    return this.processor;
  }

  connectPrimary(node) {
    this.compressor.connect(node);
  }
```

```
  connectSecondary(node) {
    this.processor.connect(node);
  }
};
```

```
let audioContext = new (window.AudioContext || window.webkitAudioContext)();

//Select two <audio> elements on the page. Ideally they have the autoplay attribute
let musicElement = document.getElementById("music");
let voiceoverElement = document.getElementById("voiceover");

//We create source nodes from them
let musicSourceNode = audioContext.createMediaElementSource(musicElement);
let voiceoverSourceNode = audioContext.createMediaElementSource(voiceoverElement);

let duckingNode = new DuckingNode(audioContext);

//Connect everything up
musicSourceNode.connect(duckingNode.primaryInput);
voiceoverSourceNode.connect(duckingNode.secondaryInput);
duckingNode.connectPrimary(audioContext.destination);
duckingNode.connectSecondary(audioContext.destination);
```

Parte de este ejemplo proviene de esta respuesta de Kevin Ennis y el código en mdn .

Preparar

Comenzamos creando un contexto de audio y luego creamos un oscilador, que es la forma más fácil de verificar que la configuración funciona. (Ejemplo de violín)

```
// We can either use the standard AudioContext or the webkitAudioContext (Safari)
var audioContext = (window.AudioContext || window.webkitAudioContext);

// We create a context from which we will create all web audio nodes
var context = new audioContext();

// Create an oscillator and make some noise
var osc = context.createOscillator();

// set a frequecy, in this case 440Hz which is an A note
osc.frequency.value = 440;

// connect the oscillator to the context destination (which routes to your speakers)
osc.connect(context.destination);

// start the sound right away
osc.start(context.currentTime);

// stop the sound in a second
osc.stop(context.currentTime + 1);
```

Lea Empezando con web-audio en línea: https://riptutorial.com/es/web-audio/topic/6984/empezando-con-web-audio

Creditos

S. No	Capítulos	Contributors
1	Empezando con web-audio	CodingIntrigue, Community, Mike McCaughan, Mikhail, Oskar Eriksson, SirPython, Sumurai8, thanksd