Playing Audio Resources Simultaneously in JavaScript

An Example From FlippyTime

Tim Cotten · Follow
Published in Cotten.IO
3 min read · Dec 10, 2018

Listen Share

This article covers the problem web browsers have in playing the same audio resource multiple times and a mitigation in JavaScript that makes it possible to "stack" multiple sound effects on top of each other.

While working on FlippyTime I found a very satisfying sound file for "swishing" the tiles.

However, when I linked the JavaScript audio play function to the sound effect to touching a tile (there might be a grid of 4x4 tiles on the screen, for instance) it would *not play* a subsequent sound until the first piece of audio had finished playing.

As you can imagine, this created an incredibly annoying and broken-sounding experience.

There are many libraries dedicated to solving this problem, but I was writing FlippyTime in VanillaJS as an experiment in pure JavaScript, and looked for a simple, effective way to do so with minimum code.

Remembering my old love of MIDI I thought "why don't I try making virtual channels, and pre-loading the same sound effect in each, then I can rotate rapidly between them when I need to play the same sound effect multiple times."

You can find the full code here in the FlippyTime main.js file, but the breakdown is straightforward:

```
function Channel(audio_uri) {
        this.audio_uri = audio_uri;
        this.resource = new Audio(audio_uri);
}

Channel.prototype.play = function() {
        // Try refreshing the resource altogether
        this.resource.play();
}
```

The `Channel` object is responsible for loading a given Audio resource. Really that's all: it's a dumb wrapper.

```
function Switcher(audio_uri, num) {
        this.channels = [];
        this.num = num;
        this.index = 0;

        for (var i = 0; i < num; i++) {
                this.channels.push(new Channel(audio_uri));
        }
}

Switcher.prototype.play = function() {
        this.channels[this.index++].play();
        this.index = this.index < this.num ? this.index : 0;
}
```

Switcher, on the other hand, loads `num` instances of the channels and loads the new Audio in each.

When `play()` is called on an instance of `Switcher` it simply rotates through the available channels and plays that particular audio stream. This allows the same audio file to be used multiple times and play simultaneously.

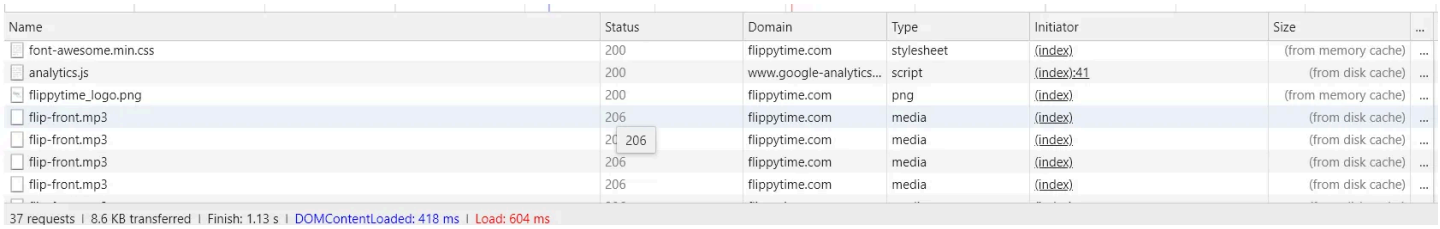
A module based implentation of the `GAME.Sound` object is below:

```
GAME.Sound = (function() {
        var self = {};

        self.playFront = function() {
                if (GAME.isReady()) { sfx_switcher_front.play(); }
        }
```

```
        self.playBack = function() {
                if (GAME.isReady()) { sfx_switcher_back.play(); }
        }

        self.playSuccess = function() {
                if (GAME.isReady()) { sfx_switcher_success.play();
}
        }

        self.playStart = function() {
                if (GAME.isReady()) { sfx_switcher_start.play(); }
        }

        self.init = function() {
                sfx_switcher_front   = new Switcher('sfx/flip-
front.mp3', 10);
                sfx_switcher_back    = new Switcher('sfx/flip-
back.mp3', 10);
                sfx_switcher_success = new
Switcher('sfx/success.mp3', 2);
                sfx_switcher_start   = new
Switcher('sfx/start.mp3', 1);
        }

        return self;
}());
```

Notice how the init loads all the needed resources ahead of time?

Keep in mind that despite reloading the same resource multiple times it's actually pretty fast thanks to the disk cache:



Summary

Audio files can be played simultaneously in JavaScript without any additional libraries simply by creating virtual channels and rotating between them.

Such a solution is fast, simple, and comes with a very small bytesize compared to the bigger game sound management libraries.

JavaScript | Game Development | Game Design | Web Development | Audio