JavaScript による音声生成

いで@いで庵
http://www.usamimi.info/~ide/

JavaScript ではサウンドまわりはこれまでろくに API が用意されていない状態が続いてきた[*1]。HTML5 で audio 要素が導入されたことで音声ファイルの再生に関する API が整ってきた[*2]が、動的に音声を生成する仕組みは用意されていなかった。

しかし、ここ一年ほどで状況が変わりつつある。いくつかのブラウザで音声生成のための API が提供されたのだ。導入された API についてここで紹介していく。

1 Audio Data API

1.1 概要

Audio Data API[*3]は Firefox4 で導入された API で、audio 要素で読み込んだ音声ファイルの波形データを読んだり、JavaScript で生成した波形データを再生することができる。Audio Data API について日本語で解説している記事[*4]もあり、割と学習は容易だと思う。

難点としては現時点[*5]で Firefox4 でしか動かないことが挙げられる。

1.2 使い方

Audio Data API による音声生成は以下の手順で行う。

1. Audio オブジェクト生成
2. Audio.mozSetup() メソッドで初期化
3. 定期的に Audio.mozWriteAudio() メソッドで波形データを書きこむ

Audio オブジェクトの生成は単に引数を渡さずに new すればよい。

```
var audio_output = new Audio();
```

Audio.mozSetup() には引数としてチャンネル数とサンプリング周波数を渡す。チャンネル数はモノラルなら 1, ステレオなら 2 を指定する。サンプリング周波数は 44100 や 48000 が一般的。

```
audio_output.mozSetup(1,44100); //チャンネル数 1 でサンプリング周波数 44.1kHz
```

Audio.mozWriteAudio() は引数として波形データを普通の配列か Float32Array で渡す。配列の各要素の値は実数で指定する。Audio.mozWriteAudio() で波形データをすべて書き込めるとは限らない。なので戻り値として書き込んだ波形データの長さを返す。

```
var sample = new Float32Array(4096);
for(var i=0;i<sample.length;i++){ //なんかてきとーに波形データ作る
  sample[i] = Math.sin(2*Math.PI*i/4096);
```

[*1] 一応 bgsound 要素や embed 要素を経由して制御することはできたが、互換性やタイムラグなどの問題があった

[*2] コーデックまわりの話で色々あるみたいなんで実用はもう少し先なんだろうな・・・

[*3] https://wiki.mozilla.org/Audio_Data_API

[*4] http://ascii.jp/elem/000/000/564/564098/

[*5] 2011 年 6 月 16 日現在

```
}
var written_length = audio_output.mozWriteAudio(sample); //波形データを書きこむ。
```

音を鳴らせ続けるには定期的に波形データを書きこめば良い。これには setTimeout() や setInterval() を使う。

1.3 サンプル (正弦波の生成)

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8" />
    <title>sample</title>
  </head>
  <body>
    <script>
      var audioDataDestination = function(generator,sample_rate){
        var audio = new Audio(),
            buffer_size = sample_rate/2, //前もって用意しておく波形データの長さ。値は適宜調整すべし
            current_write_position = 0,  //現在までに書き込んだ長さ
            tail = null,                 //前回書き込み切れなかったバッファ
            tail_position = 0;           //tail の書き込めた長さ
        audio.mozSetup(1,sample_rate);
        setInterval(function(){
          var buffer,       //バッファは Float32Array。
              written,      //書き込んだバッファサイズ
              available,    //書き込めそうなサイズ
              current_time; //現在の再生位置
          if(tail){ //前回書き込めなかったバッファが残っている場合
            written = audio.mozWriteAudio(tail.subarray(tail_position));
            current_write_position += written;
            tail_position += written;
            if(tail_position<tail.length){ //全部を書き込めなかったらオワタ
              return;
            }
            tail = null; //全部書いたので消しておｋ
          }
          //現在書き込めそうなサイズを求める
          current_time = audio.mozCurrentSampleOffset(); //現在の再生位置
          available = current_time + buffer_size - current_write_position;
          if(available>0){
            buffer = new Float32Array(available);
            generator(buffer); //波形データ生成
            written = audio.mozWriteAudio(buffer);
            if(written<buffer.length){ //書き込めなかったバッファは次回の処理まで取っておく
              tail = buffer;
              tail_position = written;
            }
            current_write_position += written;
          }
        },100); //100msec ごとに処理を行う。呼び出し間隔は適宜調整すべし
      };

      var current_position = 0,
          frequency = 440,
          sample_rate = 44100,
          k = frequency * 2 * Math.PI / sample_rate;
      audioDataDestination(function(buffer){ //440Hz の正弦波を生成する関数
        var i,len=buffer.length;
        for(i=0;i<len;i++){
          buffer[i] = Math.sin(k*current_position);
          current_position++;
        }
```

```
    },sample_rate); //サンプリング周波数 44.1kHz
  </script>
 </body>
</html>
```

2 Web Audio API

2.1 概要

Web Audio API[*6]は W3C Audio Group が策定を進めている API である。現在 Google Chrome の dev 版と canary 版[*7]、及び Safari の Mac 版で使うことができる[*8]。

Web Audio API では音の合成過程をノードの結合で表す。入力と出力を持つ様々な効果を持つノードをつなぎ合わせることでどの波形にどの順序でどのエフェクトを加えたかを表現する。destination ノードと呼ばれるノードへ入力された波形が最終的にスピーカーに流れる波形となる。

結合としては図 1 のように source ノード (入力なしで波形を出力するノード) と destination ノードを直結する他、図 2 のように複数の source ノードの出力にエフェクトをかけ、最終的に 1 つの出力にミックスして destination ノードに入力するような結合もできる。

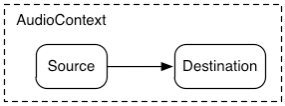

図 1　単純な結合

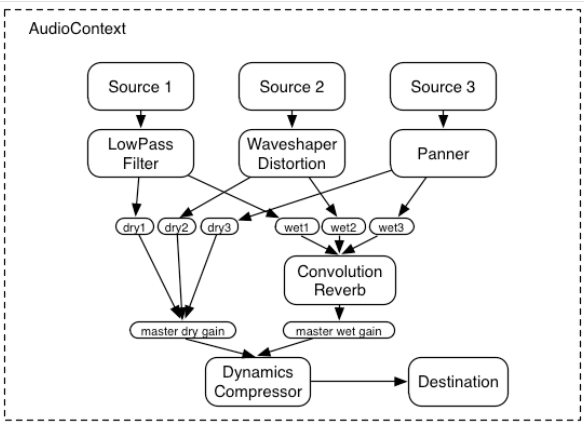

図 2　複雑な結合

2.2 使い方

Web Audio API による音声生成は以下の手順で行う[*9]。ノードの結合は先の図 1 における source ノードが JavaScriptNode に置き換わった物である。

1. AudioContext オブジェクトの生成
2. JavaScriptNode の生成と初期化
3. JavaScriptNode を AudioContext オブジェクトの destination につなぐ

AudioContext オブジェクトの生成は単に引数を渡さずに new すればよい。サンプリング周波数は AudioContext オブジェクトの sampleRate プロパティで取得できる。変更はできない。

```
var audio_context = new webkitAudioContext();
```

[*6] http://chromium.googlecode.com/svn/trunk/samples/audio/index.html
[*7] Windows の場合、Google Chrome のリリースは stable,beta,dev,canary の四種類がある
[*8] Google Chrome の場合 URL バーに about:flags と入力して出てくるページで Web Audio API を有効化する必要がある
[*9] 他にも様々なエフェクタのノードがあるが今回は JavaScript で生成した波形を再生するという目的の範囲に留めた。

```
var sample_rate = audio_context.sampleRate;
```

JavaScriptNode は onaudioprocess プロパティで指定された関数を繰り返し呼び出して波形を出力させる。

JavaScriptNode の生成は AudioContext.createJavaScriptNode() で行う。第 1 引数で onaudioprocess で一度に出力する波形の長さを指定する。指定できる長さは $2^n (n \in \{8, 9, 10, 11, 12, 13, 14\})$ のみ。第 2 引数と第 3 引数はそれぞれ入力と出力の数である。

```
var buf_size = 4096;//2 の 12 乗
var js_node = audio_context.createJavaScriptNode(buf_size,0,1);//ソースとして用いる場合入力 0 出力 1
```

JavaScriptNode.onaudioprocess には 1 引数の関数を指定する。引数として渡されるオブジェクトから出力先バッファを参照できる。このバッファに波形を書きこめば良い。

```
js_node.onaudioprocess = function(e){
  var output_l = e.outputBuffer.getChannelData(0),//ステレオの左側の出力先
      output_r = e.outputBuffer.getChannelData(1);//ステレオの右側の出力先
  var sample = new Float32Array(buf_size);
  for(var i=0;i<sample.length;i++){ //なんかてきとーに波形データ作る
    sample[i] = Math.sin(2*Math.PI*i/buf_size);
  }
  output_l.set(sample); // 出力先に波形を書きこむ
  output_r.set(sample);
};
```

最後に生成した JavaScriptNode を destination ノードにつなぐ。

```
js_node.connect(audio_context.destination);
```

2.3 サンプル (正弦波の生成)[*10]

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8" />
    <title>sample</title>
  </head>
  <body>
    <script>
      var js_node;//JavaScriptNode を格納する変数。諸事情によりグローバル変数として宣言する
      var audioDataDestination = function(generator){
        var context = new webkitAudioContext();
        js_node = context.createJavaScriptNode(4096,0,2);
        js_node.onaudioprocess = function(e){
          var output_l = e.outputBuffer.getChannelData(0),
              output_r = e.outputBuffer.getChannelData(1),
              sample = new Float32Array(4096);
          generator(sample,context.sampleRate);//波形生成
          output_l.set(sample); // 出力先に波形を書きこむ
          output_r.set(sample);
        };
        js_node.connect(context.destination);
      };

      var current_position = 0,
          frequency = 440;
      audioDataDestination(function(buffer,sample_rate){ //440Hz の正弦波を生成する関数
        var i,len=buffer.length,
```

[*10] js_node をグローバル変数として宣言している理由はローカル変数として宣言すると何故か挙動が不安定になるから (Google Chrome にて)。ガベージコレクションまわりのバグ？
関連してそうなバグ http://code.google.com/p/chromium/issues/detail?id=82795

```
            k = frequency * 2 * Math.PI / sample_rate;
          for(i=0;i<len;i++){
            buffer[i] = Math.sin(k*current_position);
            current_position++;
          }
        });
      </script>
    </body>
</html>
```

3 まとめ

Audio Data API と Web Audio API では動的音声生成に関する考えが異なっている。Audio Data API はプログラムを書く側がタイミングを見計らって波形出力するのに対し、Web Audio API は API 側がタイミングを決めて波形出力の関数を呼び出している。エフェクタの結合で音声合成を行うという Web Audio API の考えもそうだが、この 2 つの API は API 側がどれだけ世話をしてくれるかにはっきりと違いがある。

実際に音声生成を JavaScript で行う場合にはより多くのブラウザで動くようにするために両方の API を利用することになると思うが、それぞれの API の考えが違うと API のラッパを作るのが難しい。こと両者の API の強みを活かすということは難しく、どうにも両者の API の最小公倍数的なものになってしまう。

とはいえ、これまでの JavaScript のサウンドまわりの貧弱さからするとこれらの API の登場はかなりの進歩であり、今後の JavaScript コンテンツの成長が期待される。

4 おまけ：既存の方法について

これまで説明してきた API はここ 1 年の間に生まれたものであるが、それ以前に存在した動的な音生成の方法を紹介していく。

Flash と連携する方法 JavaScript と Flash(ActionScript) を連携させ、Flash10 の動的音生成の機能を JavaScript から使う。ブラウザ互換性が良い一方で、結局 Flash に仕事を投げていることへの後ろめたさから開発モチベーションが低下するといった短所がある。

audio 要素と dataURI を組み合わせる方法 波形データを wave ファイル等の形式で表現し、そのファイルを dataURI に変換して audio 要素の src に流しこんで再生する。波形データを生成しながら再生するといったことができないのが難点。