MỤC LỤC

Chương 11. Ứng dụng web audio phần 1 5
11.1 Vị trí của chương trong cuốn sách 5
11.2 Đồ thị node — mô hình lập trình của Web Audio 8
11.3 Ngữ cảnh, đồng hồ và vòng đời node 13
11.4 Nguồn phát 16
11.5 Xử lý thời gian thực — trễ và ngân sách 19
11.6 AudioWorklet — mang mã riêng vào 24
11.7 Lập lịch sự kiện và đường bao tham số 28
11.8 Đo và gỡ lỗi trong trình duyệt 33
11.9 Thực hành, bài tập và tổng kết 35

DANH MỤC HÌNH VẼ

Hình 11.1 Bảy mục kỹ thuật của chương 11.....6
Hình 11.2 Hai kiểu chạy, và hai cách viết cùng một chuỗi xử lý.....6
Hình 11.3 Một đồ thị node gồm sáu khối của cả cuốn sách.....9
Hình 11.4 Bảng đối chiếu node dựng sẵn với mục đã học.....10
Hình 11.5 Tám nguồn cùng gọi start(0), và con số ấy khớp công thức nào.....11
Hình 11.6 Kết quả thật của đồ thị, và ba đường đáp ứng chồng khít nhau.....12
Hình 11.7 Hai loại ngữ cảnh và luật cử chỉ người dùng.....13
Hình 11.8 Đồng hồ nhảy từng khối, nhưng lịch thì chính xác tới mẫu.....14
Hình 11.9 Bốn trạng thái của một node nguồn.....15
Hình 11.10 Bốn dạng sóng dựng sẵn, và biên độ các hoạ ba của chúng.....17
Hình 11.11 Răng cưa 220 Hz và 5 kHz: tự tính so với OscillatorNode.....17
Hình 11.12 Cùng một bộ đệm, ba cách phát, và hai đại lượng đo được.....18
Hình 11.13 Ba bên trong một vòng lặp thời gian thực.....19
Hình 11.14 Độ trễ theo kích thước khối, và ba chỗ trễ.....20
Hình 11.15 Tải trọng theo số giọng, và cách suy ra số giọng chạy nổi.....22
Hình 11.16 Ba khối bị trả muộn, và dấu vết của chúng trong phổ.....23
Hình 11.17 Hai luồng, và hai cách truyền số qua ranh giới.....24
Hình 11.18 Bản Python và bản AudioWorklet, và hiệu của chúng.....26
Hình 11.19 Cùng một đường dốc, hai kiểu tham số, và chênh lệch giữa chúng.....27
Hình 11.20 Bảy điều cấm trong luồng âm thanh.....28
Hình 11.21 Ba kiểu đường bao đo trên đầu ra thật, và cách chọn.....29
Hình 11.22 Hạ gain bằng ba cách khác nhau, và tiếng tách đo được.....30
Hình 11.23 Bộ lập lịch cửa sổ nhìn trước và kết quả đo được.....32
Hình 11.24 Phổ lấy thẳng từ AnalyserNode, và ba con số phải đặt đúng.....34
Hình 11.25 Bảy lỗi hay gặp và cách kiểm từng lỗi.....35
Hình 11.26 Sáu câu phải trả lời được sau chương 11.....37

DANH MỤC BẢNG BIỂU

Bảng 11.1 Bảy mục và thứ mà mỗi mục dựng trên....7
Bảng 11.2 Ba thứ dễ gộp nhầm vào chương 11....8
Bảng 11.3 Đo lại trên đồ thị chạy thật....11
Bảng 11.4 Ba nguồn số, và chúng khớp nhau tới đâu....12
Bảng 11.5 Ba đồng hồ trong một trang web, và chúng khác nhau thế nào....15
Bảng 11.6 Ba loại nguồn phát của Web Audio....16
Bảng 11.7 Năng lượng nằm NGOÀI mọi hoạ ba của tần số đang chơi....18
Bảng 11.8 Đo lại trên ba bản phát cùng một bộ đệm....19
Bảng 11.9 Độ trễ theo kích thước khối của card, ở 44,1 kHz....21
Bảng 11.10 Đo trên một sóng sin 300 Hz mất ba khối trong 60 ms....23
Bảng 11.11 Kết quả phép kiểm chéo....26
Bảng 11.12 Hai chế độ tham số....27
Bảng 11.13 Ba cách hạ gain từ 0,8 xuống 0,08 tại mốc 250 ms....30
Bảng 11.14 Hai cách đặt lịch, đo trên cùng mười sáu nốt....33
Bảng 11.15 Ba con số cấu hình của AnalyserNode....34
Bảng 11.16 Bảy bài tập của chương 11....36
Bảng 11.17 Những thứ mới của chương 11 và chỗ dùng lại....38

CHƯƠNG 11
ỨNG DỤNG WEB AUDIO PHẦN 1

Mục tiêu của chương

Sau chương này, người đọc phải làm được những việc sau.

- Dựng được một đồ thị node hoàn chỉnh và giải thích được từng khối trong đó tương ứng mục nào đã học.
- Chứng minh được bằng số rằng node dựng sẵn của trình duyệt và mô-đun tự viết cho cùng kết quả.
- Phân biệt được ba đồng hồ trong một trang web, và nói được đồng hồ nào dùng để đặt sự kiện phát tiếng.
- Giải thích được vì sao một node nguồn chỉ dùng được một lần, và viết đúng vòng đời của nó.
- Tính được độ trễ đường vòng và ngân sách tính toán, và *đo* được cả hai trên máy của người đọc.
- Viết được một AudioWorkletProcessor, và kiểm chéo nó với bản Python cùng thuật toán.
- Chọn đúng giữa a-rate và k-rate, và đọc tham số theo mẫu chuẩn.
- Đổi một tham số mà không gây tiếng tách, và *đo* được mức tiếng tách đã giảm.
- Cài được bộ lập lịch cửa sổ nhìn trước và chọn có căn cứ hai con số của nó.
- Gỡ được bảy lỗi hay gặp bằng cách *đo* chứ không bằng cách nghe ngóng.

Một chương không thêm phép xử lý nào

Đọc danh sách node của Web Audio thì thấy đúng danh sách những thứ đã tự viết từ chương 5 tới chương 10. Mục 11.2.4 đo lại và cho thấy chúng khớp nhau tới 0,001 decibel.

Cái mới của chương này là *quyền điều khiển*. Mười chương qua, chương trình của người đọc gọi hàm xử lý khi nào nó muốn. Từ đây, trình duyệt gọi mã của chúng ta cứ 2,90 mili giây một lần, và trả muộn một lần là người nghe nghe thấy.

Bốn mục có phép kiểm bằng số nên tự chạy lại: đáp ứng biquad ở mục 11.2.4, độ chính xác của lịch ở mục 11.3.2, kiểm chéo bộ nén ở mục 11.6.2, và tiếng tách ở mục 11.7.2.

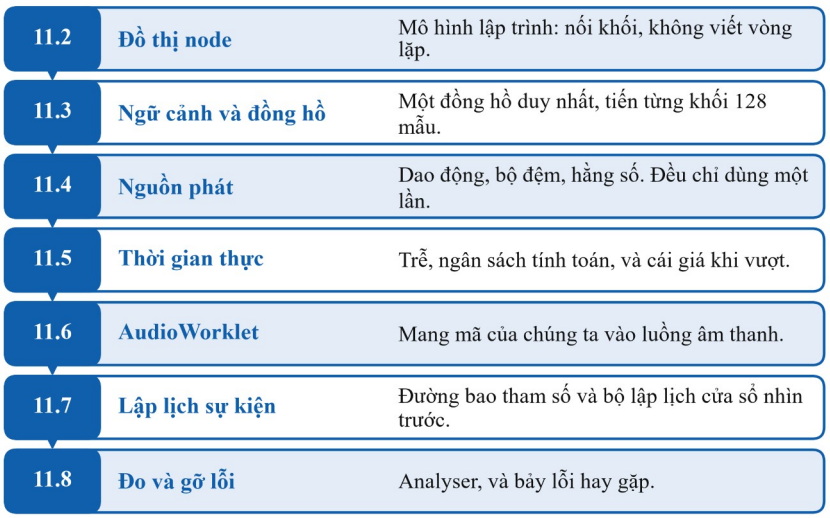
11.1 Vị trí của chương trong cuốn sách

11.1.1 Mười chương vừa qua, chương trình của chúng ta cầm quyền

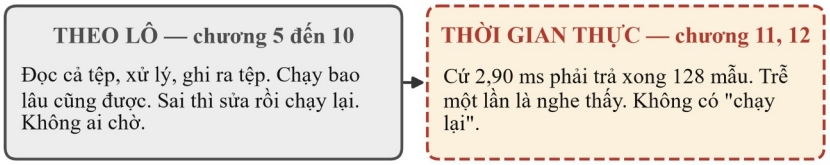
Từ chương 5 tới chương 10, mọi chương trình đã viết đều có cùng một hình dạng: đọc cả tệp vào bộ nhớ, xử lý, ghi kết quả ra tệp. Chương trình quyết định khi

nào gọi hàm nào. Nếu một phép tính chạy mất mười giây thì cũng không sao — không ai chờ.

Chương này đổi hẳn luật chơi ấy. Từ đây, quyền điều khiển thuộc về trình duyệt: nó gọi mã của chúng ta cứ 2,90 mili giây một lần, và mỗi lần chúng ta phải trả về đúng 128 mẫu. Trả muộn một lần thì người nghe nghe thấy một khoảng lặng.

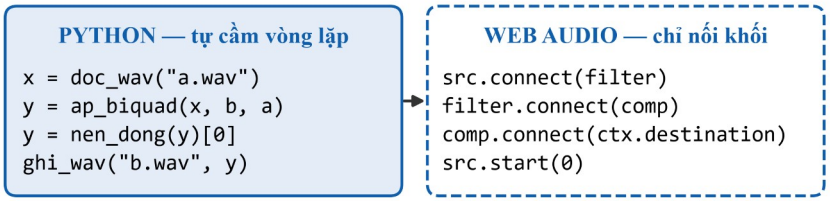


Hình 11.1 Bảy mục kỹ thuật của chương 11



Cùng những phép xử lý ấy, nhưng bị đặt vào một ràng buộc thời gian không thương lượng được.

(a) Hai kiểu chạy



Đoạn mã bên phải không có vòng lặp nào. Vòng lặp nằm trong trình duyệt, chạy ở một luồng khác, và gọi mã của chúng ta.

(b) Cùng một chuỗi xử lý, hai cách viết

Hình 11.2 Hai kiểu chạy, và hai cách viết cùng một chuỗi xử lý

11.1.2 Điều đáng ngạc nhiên: chương này không thêm phép xử lý nào

Đọc lướt tên các node của Web Audio thì thấy ngay: GainNode, DelayNode, BiquadFilterNode, DynamicsCompressorNode, ConvolverNode, StereoPannerNode, PannerNode. Đó đúng là danh sách những thứ đã tự viết bằng Python từ chương 5 tới chương 10.

Và chúng không chỉ giống tên. Mục 11.2.4 đo lại đáp ứng của BiquadFilterNode trên tiếng ra thật, rồi so với công thức RBJ đã cài ở mục 9.3: hai đường lệch nhau trung bình 0,001 decibel. Chúng là một thứ.

Vậy chương này học cái gì

Học cách một hệ thống âm thanh *thời gian thực* được tổ chức: ai gọi ai, đồng hồ nào là đồng hồ thật, và mỗi lần được gọi thì có bao nhiêu thời gian.

Học cách đặt một sự kiện vào đúng thời điểm — chính xác tới một mẫu — trong khi ngôn ngữ đang dùng chỉ hẹn giờ được với sai số vài mili giây.

Và học cách mang chính những hàm đã viết bằng Python vào chạy trong trình duyệt, rồi *chứng minh* được rằng chúng cho cùng kết quả.

11.1.3 Mọi con số trong chương này đều là số đo

Chương này có một điểm khác các chương trước về cách dựng tài liệu. Toàn bộ mã JavaScript trong bài đã được *chạy thật* trên một bản cài đặt Web Audio đầy đủ, kết quả đổ ra tệp, rồi đo lại bằng chính mô-đun Python của các chương trước.

Bảng 11.1 Bảy mục và thứ mà mỗi mục dựng trên

Mục	Nội dung	Dựng trên
11.2	Đồ thị node — mô hình lập trình	Toàn bộ các chương 5 tới 10; và headroom của mục 10.2
11.3	Ngữ cảnh, đồng hồ và vòng đời node	Hoàn toàn mới
11.4	Nguồn phát: dao động, bộ đệm, hằng số	Dạng sóng của 2.5; chồng phổ của 3.4; cent của 2.6
11.5	Thời gian thực: trễ, ngân sách, đứt tiếng	Xử lý theo khối của 6.8; tiếng tách của 6.6
11.6	AudioWorklet — mang mã riêng vào	Mọi thuật toán đã viết, đặc biệt bộ nén của 8.3
11.7	Lập lịch sự kiện và đường bao tham số	Fade của 6.6; bộ làm trơn một cực của 8.2
11.8	Đo và gỡ lỗi trong trình duyệt	Phổ và cửa sổ của 4.5 và 4.6

11.1.4 Những gì chương này không làm

Bảng 11.2 Ba thứ dễ gộp nhầm vào chương 11

Thứ dễ gộp nhầm	Vì sao không thuộc chương này	Thuộc chương nào
Rack hiệu ứng dựng được từ giao diện, và trộn động theo sự kiện	Chúng cần đủ cả bảy mục của chương này làm nền, và cần thêm cách tổ chức mã cho một ứng dụng thật	Chương 12
Tối ưu sâu: SIMD, WebAssembly, SharedArrayBuffer	Chỉ đáng làm sau khi đã *đo* và biết chỗ nào chậm. Chương này dừng ở chỗ đo được ngân sách	Chương 12
Thu tiếng từ micro và xử lý trực tiếp	Cần thêm phần quyền truy cập thiết bị và xử lý luồng phương tiện, vốn không thuộc phần âm thanh	Ngoài phạm vi sách

Ngôn ngữ của hai chương này

Từ chương này, mã trong bài viết bằng JavaScript chứ không bằng Python. Lý do đơn giản: Web Audio là một giao diện lập trình của trình duyệt, và trình duyệt chỉ chạy JavaScript.

Người đọc không cần biết JavaScript từ trước. Phần cú pháp thật sự dùng trong hai chương chỉ gồm: khai báo biến, hàm, mảng, vòng lặp for, và lớp. Tất cả đều gần với Python tới mức đọc là hiểu.

Ba khác biệt phải nhớ: JavaScript kết thúc câu lệnh bằng dấu chấm phẩy và gói khối lệnh bằng ngoặc nhọn; số thực của nó luôn là 64 bit, nhưng mảng âm thanh thì là Float32Array — 32 bit, đúng như mục 3.6; và nó *không* có toán tử luỹ thừa hai dấu sao cho mọi trường hợp, nên bài dùng Math.pow cho rõ ràng.

11.2 Đồ thị node — mô hình lập trình của Web Audio

11.2.1 Không viết vòng lặp, chỉ nối khối

Trong Python, muốn lọc một tín hiệu thì gọi một hàm nhận mảng vào và trả mảng ra. Trong Web Audio thì không: chúng ta tạo một node lọc, nối nguồn vào nó, nối nó vào đích, rồi bảo nguồn bắt đầu. Vòng lặp nằm ở đâu đó bên trong trình duyệt, viết bằng C++, chạy ở một luồng khác.

Liệt kê 11.1 Cùng một chuỗi xử lý, viết bằng Web Audio

```
const ctx = new AudioContext();

const src = ctx.createBufferSource();
src.buffer = buf;                    // AudioBuffer đã nạp sẵn

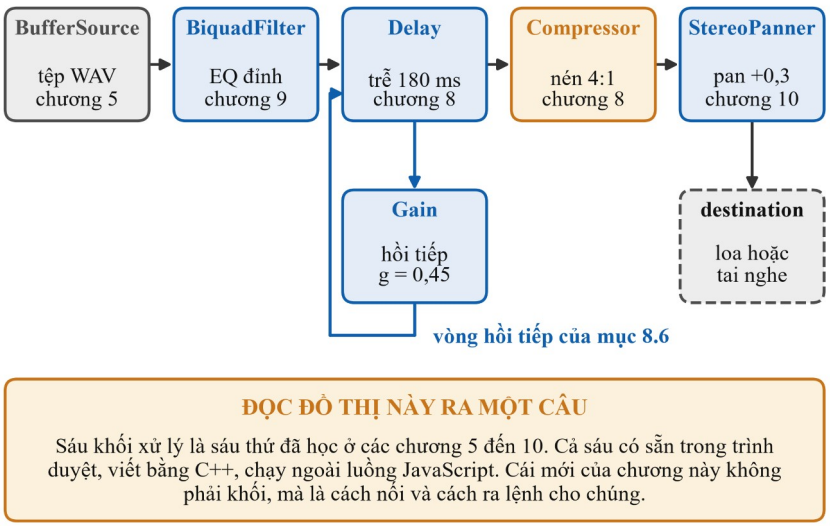
const eq = ctx.createBiquadFilter();
```

```
eq.type = 'peaking';
eq.frequency.value = 900;
eq.Q.value = 1.2;
eq.gain.value = -8;

const comp = ctx.createDynamicsCompressor();
comp.threshold.value = -18;
comp.ratio.value = 4;

src.connect(eq).connect(comp).connect(ctx.destination);
src.start(0);
```

Mười bảy dòng, và không dòng nào là một vòng lặp. Hàm connect trả về chính node đích, nên nối được thành chuỗi như dòng áp chót. Đó là toàn bộ mô hình lập trình.



Hình 11.3 Một đồ thị node gồm sáu khối của cả cuốn sách

11.2.2 Mỗi node dựng sẵn là một mục đã học

Node	Làm gì	Học ở (mục)	Còn thiếu gì
GainNode	Nhân một hệ số	5.4 — gain theo dB	Không thiếu gì
DelayNode	Trễ tới 180 giây	6.4, 8.5 — trễ, lọc lược	Không nội suy bậc cao
BiquadFilterNode	Bảy kiểu lọc bậc hai	9.3 — công thức RBJ	Không đặt thẳng được hệ số
IIRFilterNode	Lọc bất kỳ theo b và a	9.3 — sai phân	Hệ số cố định, không tự động hoá
DynamicsCompressorNode	Bộ nén động	8.3 — nén và giới hạn	Không có bộ giới hạn riêng
ConvolverNode	Tích chập đáp ứng xung	9.8 — vang tích chập	Tự chuẩn hoá, hay gây bất ngờ
StereoPannerNode	Pan hai kênh	10.4 — giữ công suất	Chỉ một luật, không chọn được
PannerNode	Đặt nguồn trong không gian	10.6, 10.7 — ITD, ILD	Bảng HRTF của trình duyệt
AnalyserNode	FFT theo khung	4.5 — phổ và cửa sổ	Chỉ để nhìn, không xử lý
AudioWorkletNode	Mã của chính mình	Mọi thứ viết bằng Python	Đây là chỗ bù mọi chỗ thiếu ở trên

Đọc cột thứ ba theo chiều dọc: đó gần như là mục lục của mười chương vừa qua.

Hình 11.4 Bảng đối chiếu node dựng sẵn với mục đã học

Cột thứ ba của bảng đọc theo chiều dọc gần như là mục lục của mười chương vừa qua. Cột thứ tư quan trọng không kém: nó nêu chỗ node dựng sẵn *thiếu* so với bản tự viết, và đó chính là danh sách lý do người ta phải dùng AudioWorklet ở mục 11.6.

Ba chỗ thiếu đáng chú ý nhất

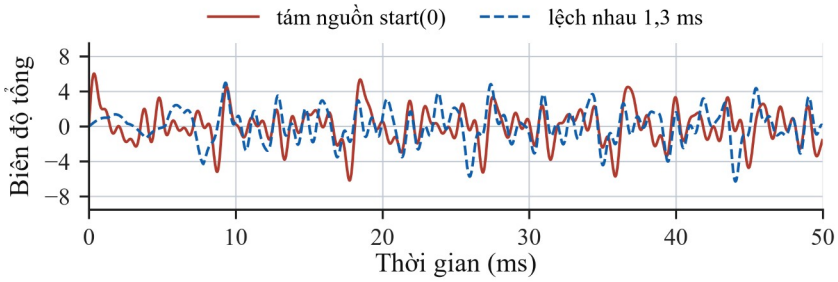
BiquadFilterNode không cho đặt thẳng năm hệ số. Muốn cài một bộ lọc thiết kế riêng thì phải dùng IIRFilterNode — nhưng node ấy lại *không* cho đổi hệ số sau khi tạo, nên không làm wah của mục 9.5 được.

StereoPannerNode chỉ có một luật pan duy nhất, và đó là luật giữ công suất của mục 10.4.3. Muốn luật thoả hiệp thì phải tự dựng bằng hai GainNode.

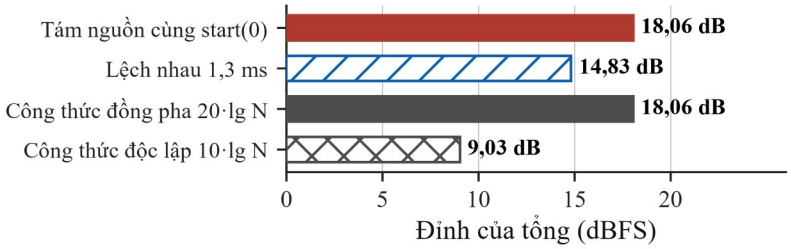
Không có bộ giới hạn riêng. DynamicsCompressorNode đặt tỉ số 20 là hết cỡ, và nó không có lookahead của mục 8.4 — nên không bảo đảm được trần.

11.2.3 Cộng nhiều nhánh, và đồ thị không tự bảo vệ

Nối nhiều node vào cùng một node đích thì chúng tự cộng lại. Đây là chỗ headroom của mục 10.2 quay lại, và nó quay lại theo cách bất tiện nhất.



(a) Tổng tám dao động hình sin



(b) Đỉnh đo được và công thức

Hình 11.5 Tám nguồn cùng gọi start(0), và con số ấy khớp công thức nào

Tám dao động hình sin ở tám tần số khác nhau, mỗi cái nối thẳng vào một GainNode có hệ số 1. Trực giác nói tám nguồn *khác* tần số thì độc lập, nên đỉnh phải tăng chừng mười lần loga của tám, tức 9,03 decibel.

Bảng 11.3 Đo lại trên đồ thị chạy thật

Cách làm	Đỉnh đo được	Khớp công thức nào
Tám nguồn cùng gọi start(0)	+18,06 dBFS	Công thức *đồng pha* của mục 10.2.3: hai mươi lần loga của tám bằng 18,06
Tám nguồn khởi động lệch nhau 1,3 ms	+14,83 dBFS	Nằm giữa hai công thức — và vẫn cao hơn công thức độc lập nhiều
Công thức độc lập, để so sánh	+9,03 dBFS	Đây là con số mà trực giác đưa ra, và nó *sai* trong cả hai trường hợp trên

Vì sao tám tần số khác nhau lại đồng pha

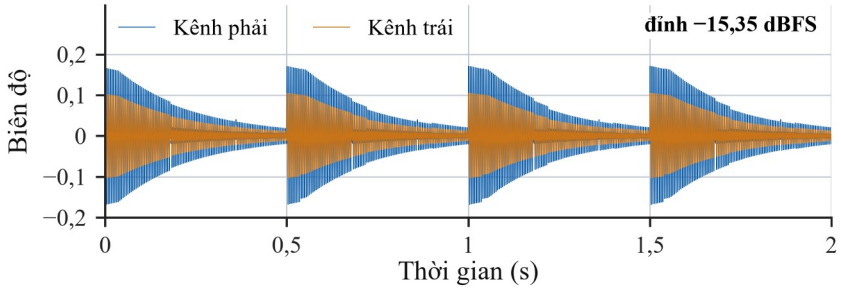
Vì tất cả đều bắt đầu ở cùng một thời điểm với cùng pha ban đầu bằng 0. Ngay tại thời điểm ấy và ở những thời điểm gần đó, đạo hàm của chúng cùng dấu, nên đỉnh của chúng chồng lên nhau.

Hệ số tương quan tính trên cả đoạn dài thì gần 0 — nhưng *đỉnh* thì không quan tâm tới cả đoạn dài. Nó chỉ cần một mẫu duy nhất mà tám sóng cùng lớn.

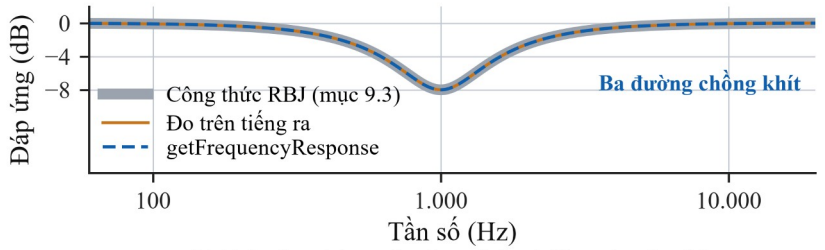
Cách chữa rẻ nhất: cho mỗi nguồn khởi động lệch nhau một chút, hoặc đặt pha ban đầu ngẫu nhiên. Cách chắc chắn: chừa headroom theo công thức *đồng pha*, tức bi quan nhất.

11.2.4 Biquad của trình duyệt có đúng công thức của chương 9 không

Đây là phép kiểm quan trọng nhất của cả chương, vì nếu câu trả lời là "không" thì mọi hiểu biết của mười chương trước không mang sang đây được. Phép kiểm dùng ba nguồn số độc lập nhau.



(a) Kết quả thật của đồ thị ở hình trước



(b) Đáp ứng biquad: công thức, hỏi node, đo thật

Hình 11.6 Kết quả thật của đồ thị, và ba đường đáp ứng chồng khít nhau

Bảng 11.4 Ba nguồn số, và chúng khớp nhau tới đâu

Nguồn số	Lấy ở đâu	Lệch so với công thức
Công thức RBJ	Hàm biquad_he_so và dap_ung_biquad_db của mục 9.3, viết bằng Python	Đây là mốc để so
getFrequencyResponse	Hỏi thẳng chính node của trình duyệt	0,0000 dB
Đo trên tiếng ra	Cho nhiễu trắng đi qua node, lấy hiệu hai phổ	Trung bình 0,001 dB; lớn nhất 0,055 dB

Kết luận rút ra không phải "trình duyệt cài đúng" — điều ấy vốn dĩ nhiên — mà là: mô-đun tự viết trong mười chương qua và node dựng sẵn của trình duyệt là *cùng một thứ*. Vặn núm nào trên node cũng ra hệ quả đúng như đã học, và mọi phép đo đã viết đều dùng lại được để kiểm một đồ thị Web Audio.

Một cách làm việc rút ra từ mục này

Muốn kiểm một đồ thị Web Audio mà không cần tai và không cần ngồi bấm: dựng nó bằng OfflineAudioContext, lấy AudioBuffer trả về, ghi ra tệp, rồi đo bằng đúng những hàm đã viết ở các chương trước.

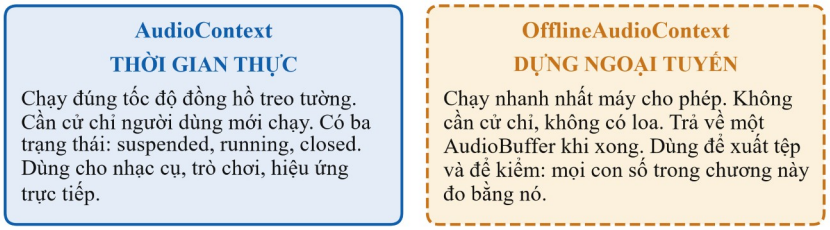
Cách này chạy nhanh hơn thời gian thực rất nhiều lần, nên chạy được hàng trăm lần trong một vòng kiểm tự động.

Toàn bộ số liệu của chương này được dựng đúng như thế.

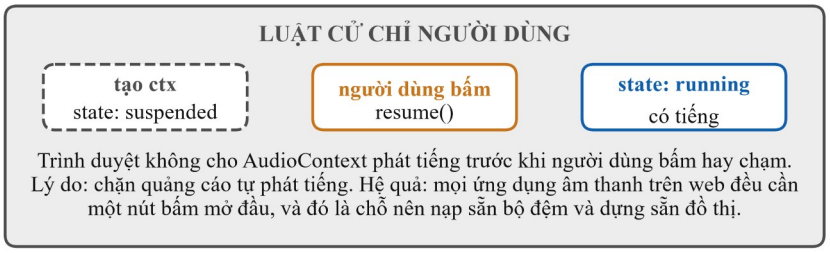
11.3 Ngữ cảnh, đồng hồ và vòng đời node

11.3.1 Hai loại ngữ cảnh, và chọn cái nào

Mọi thứ trong Web Audio sống bên trong một ngữ cảnh âm thanh. Có đúng hai loại, và chọn sai loại là sai ngay từ dòng đầu.



(a) Hai loại ngữ cảnh



(b) Luật cử chỉ người dùng

Hình 11.7 Hai loại ngữ cảnh và luật cử chỉ người dùng

Liệt kê 11.2 Tạo ngữ cảnh, và luật cử chỉ người dùng

```
// Ngữ cảnh thời gian thực — phải mở bằng một cú bấm
const ctx = new AudioContext();
document.querySelector('#batdau').addEventListener('click',
  async () => {
    await ctx.resume();          // BẮT BUỘC ở đây, không ở nơi khác
    console.log(ctx.state);      // 'running'
  });

// Ngữ cảnh ngoại tuyến — dựng nhanh hết mức, không cần cử chỉ
const off = new OfflineAudioContext(2, 44100 * 5, 44100);
// ... dựng đồ thị ở đây ...
const buf = await off.startRendering();   // trả về AudioBuffer
```

Luật cử chỉ người dùng — bất tiện nhưng có lý do

Trình duyệt tạo AudioContext ở trạng thái suspended và không cho nó phát tiếng cho tới khi người dùng bấm hay chạm một cái gì đó. Lý do là chặn quảng cáo tự phát tiếng.

Hệ quả với người lập trình: mọi ứng dụng âm thanh trên web đều phải có một nút bấm mở đầu. Và đó là chỗ tốt nhất để nạp sẵn mọi bộ đệm, nạp module worklet, và dựng sẵn đồ thị.

Triệu chứng khi bỏ sót: không có tiếng gì cả, và không có lỗi nào. In ctx.state ra là biết ngay.

11.3.2 Đồng hồ ngữ cảnh — chính xác nhưng nhảy từng bậc

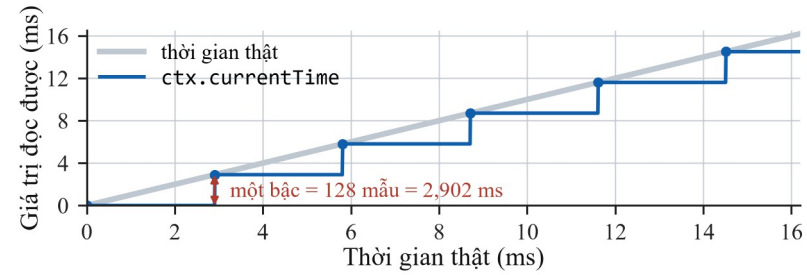
Ngữ cảnh có một đồng hồ riêng, đọc bằng ctx.currentTime, tính bằng giây kể từ khi ngữ cảnh được tạo. Đây là đồng hồ *duy nhất* được dùng để đặt thời điểm phát tiếng.

$$t_q = \frac{N_q}{f_s} = \frac{128}{44100} \approx 2{,}90 \text{ ms} \tag{11.1}$$

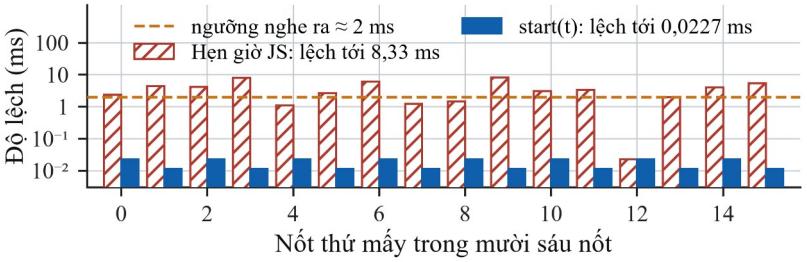
Trong đó N_q là kích thước khối dựng của Web Audio, cố định bằng 128 mẫu trong chuẩn, và f_s là tần số lấy mẫu của ngữ cảnh. Ở 44,1 kilohertz thì mỗi khối kéo dài 2,90 mili giây, và hàm xử lý bị gọi 344,53 lần mỗi giây.

Đồng hồ không chạy liên tục mà nhảy từng khối một.

$$t_k = k\frac{N_q}{f_s}, \qquad k = 0, 1, 2, \ldots \tag{11.2}$$



(a) Đồng hồ ngữ cảnh nhảy từng khối



(b) Lịch chính xác tới mẫu

Hình 11.8 Đồng hồ nhảy từng khối, nhưng lịch thì chính xác tới mẫu

Hai điều ngược nhau, và phải nắm cả hai

Con số ctx.currentTime đọc từ JavaScript chỉ đổi mỗi 128 mẫu — đo trên chính đầu ra của một worklet ra đúng 2,902 mili giây mỗi bước. Nó *không* dùng được để đặt nốt vào đúng nhịp.

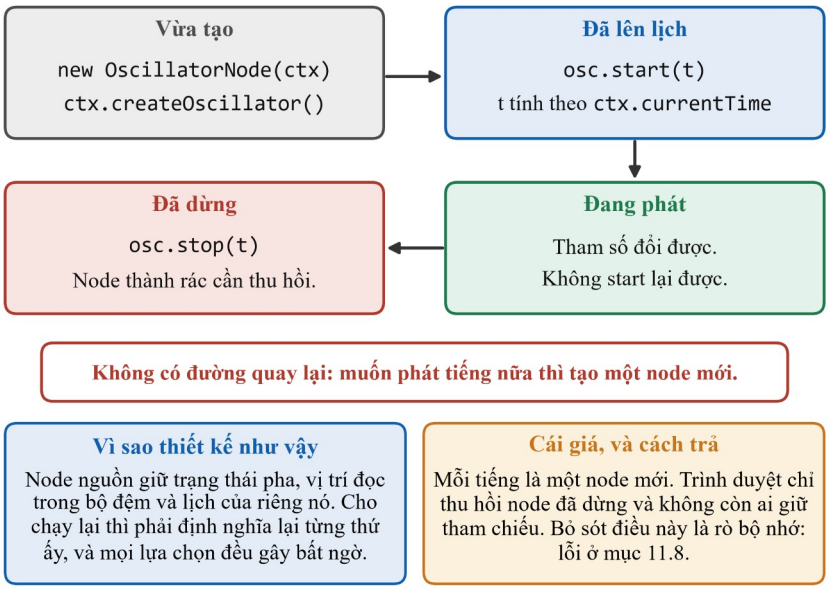
Nhưng thời điểm truyền cho start(t) lại được thực hiện chính xác tới *mẫu*. Mười sáu nốt móc kép ở 120 nhịp một phút lệch nhiều nhất 0,0227 mili giây — đúng một mẫu ở 44,1 kilohertz.

Cách dùng đúng: đọc currentTime để biết đang ở đâu, rồi cộng thêm một khoảng và truyền tổng ấy cho start. Đừng bao giờ dùng chính currentTime làm thời điểm phát.

Bảng 11.5 Ba đồng hồ trong một trang web, và chúng khác nhau thế nào

Đồng hồ	Nguồn	Dùng để làm gì
ctx.currentTime	Thạch anh của card âm thanh	*Mọi* thời điểm phát tiếng. Đây là đồng hồ thật của bài toán này
performance.now()	Bộ đếm của hệ điều hành	Đo hiệu năng, vẽ giao diện. Nó *trôi* so với đồng hồ trên
setTimeout, setInterval	Hàng đợi sự kiện của trình duyệt	Chỉ để đánh thức bộ lập lịch — xem mục 11.7.4. Sai số vài mili giây, và dừng hẳn khi tab ở nền

11.3.3 Vòng đời một node nguồn — dùng một lần rồi bỏ



Hình 11.9 Bốn trạng thái của một node nguồn

Đây là chỗ gây bất ngờ nhiều nhất cho người mới. Một node nguồn — OscillatorNode hay AudioBufferSourceNode — chỉ phát được *một* lần. Gọi start lần thứ hai trên cùng một node là lỗi.

Liệt kê 11.3 Cách đúng: mỗi tiếng là một node mới

```
function danhTrong(t) {
  const s = ctx.createBufferSource();
  s.buffer = mauTrong;              // dùng lại BỘ ĐỆM, không dùng lại node
  const g = ctx.createGain();
  g.gain.setValueAtTime(0.8, t);
  s.connect(g).connect(bus);
  s.start(t);
  s.onended = () => { s.disconnect(); g.disconnect(); };
}
```

> **Hai thứ phải phân biệt rõ**
>
> AudioBuffer là *dữ liệu* — mảng mẫu đã nạp. Nó có dung lượng lớn, nạp một lần, và dùng lại mãi mãi cho hàng nghìn tiếng.
>
> AudioBufferSourceNode là một *con trỏ đọc* vào dữ liệu ấy, cộng với lịch của riêng nó. Nó nhẹ, tạo mới cho từng tiếng, và bỏ đi sau khi dùng.
>
> Nhầm hai thứ này ra hai lỗi trái ngược: nạp lại bộ đệm cho mỗi tiếng thì tốn bộ nhớ và tốn thời gian; dùng lại node thì báo lỗi ngay ở tiếng thứ hai.

Dòng cuối của đoạn mã trên là dòng dễ bỏ sót nhất. Trình duyệt chỉ thu hồi được một node khi nó đã dừng *và* không còn ai giữ tham chiếu tới nó. Giữ mọi node đã tạo trong một mảng để "quản lý" là cách rò bộ nhớ nhanh nhất — và triệu chứng của nó là ứng dụng càng chạy càng chậm, đúng dòng thứ năm của bảng lỗi ở mục 11.8.

11.4 Nguồn phát

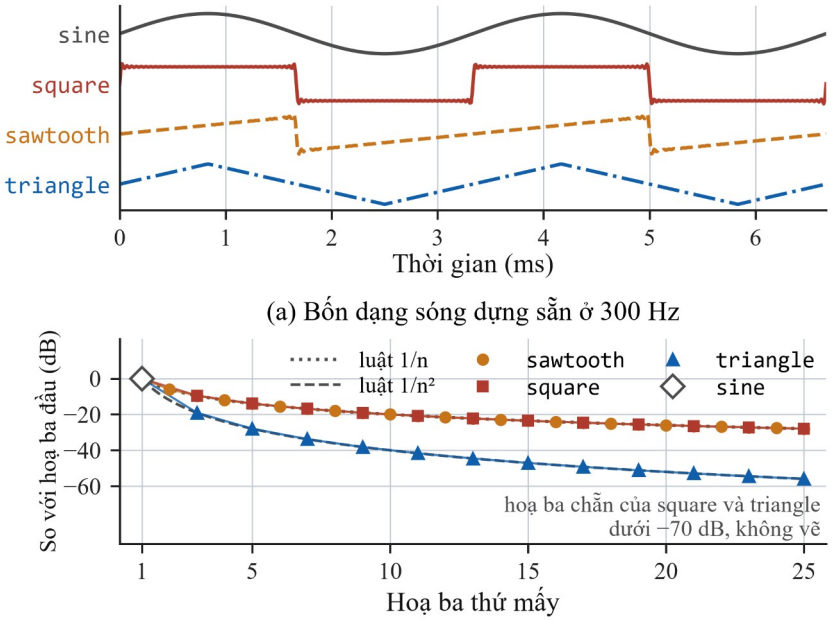
11.4.1 Ba loại nguồn, và chọn loại nào

Bảng 11.6 Ba loại nguồn phát của Web Audio

Node	Sinh ra gì	Dùng khi
OscillatorNode	Bốn dạng sóng dựng sẵn, hoặc một bảng sóng tuần hoàn tự đặt	Nhạc cụ tổng hợp, tiếng thử, sóng điều biến
AudioBufferSourceNode	Đọc lại một AudioBuffer đã nạp	Tiếng thu sẵn: trống, giọng nói, hiệu ứng của trò chơi
ConstantSourceNode	Một hằng số, nhưng là hằng số *tự động hoá được*	Điều khiển nhiều tham số cùng lúc bằng một đường bao duy nhất

Node thứ ba nghe vô dụng nhưng lại rất có ích: nối đầu ra của nó vào nhiều AudioParam khác nhau thì một đường bao điều khiển được cả nhóm. Toàn bộ số liệu về đường bao ở mục 11.7 đo bằng chính node này.

11.4.2 Dao động của trình duyệt là băng giới hạn



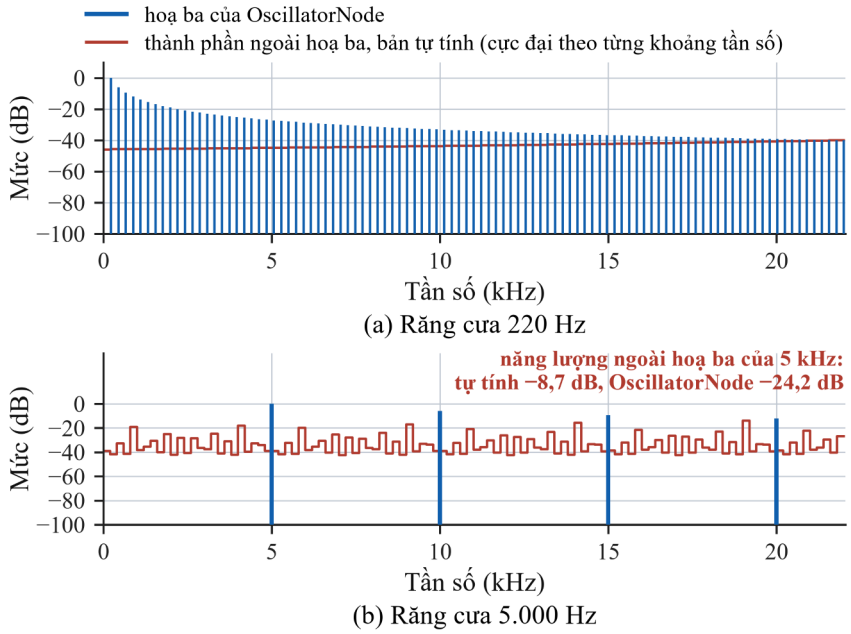
(a) Bốn dạng sóng dựng sẵn ở 300 Hz

(b) Biên độ hoạ ba so với luật lý thuyết

Hình 11.10 Bốn dạng sóng dựng sẵn, và biên độ các hoạ ba của chúng

Bốn dạng sóng nhìn thì hoàn toàn giống bốn hàm đã tự viết ở mục 2.5, và biên độ các hoạ ba đo được cũng bám đúng luật lý thuyết: răng cưa theo một trên n, sóng vuông bỏ hết hoạ ba chẵn, tam giác theo một trên n bình phương.

Nhưng chúng *không* được sinh bằng công thức. Trình duyệt dựng chúng từ một bảng sóng đã cắt bỏ mọi hoạ ba vượt quá Nyquist, rồi nội suy. Sự khác biệt ấy chỉ lộ ra ở tần số cao.



(a) Răng cưa 220 Hz

(b) Răng cưa 5.000 Hz

Hình 11.11 Răng cưa 220 Hz và 5 kHz: tự tính so với OscillatorNode

Bảng 11.7 Năng lượng nằm NGOÀI mọi hoạ ba của tần số đang chơi

Cách sinh răng cưa 5 kHz	Năng lượng lạc	Tính ra phần trăm
Tự tính bằng công thức, đúng như mục 2.5	−8,7 dB	Khoảng 13,5 phần trăm tổng năng lượng
OscillatorNode của trình duyệt	−24,2 dB	Khoảng 0,38 phần trăm — nhỏ hơn ba mươi lần

Vì sao điều này quan trọng với người làm nhạc

Chỗ năng lượng lạc ấy là các hoạ ba trên Nyquist gập ngược lại — đúng hiện tượng của mục 3.4. Chúng rơi vào những tần số không liên quan gì tới cao độ đang chơi.

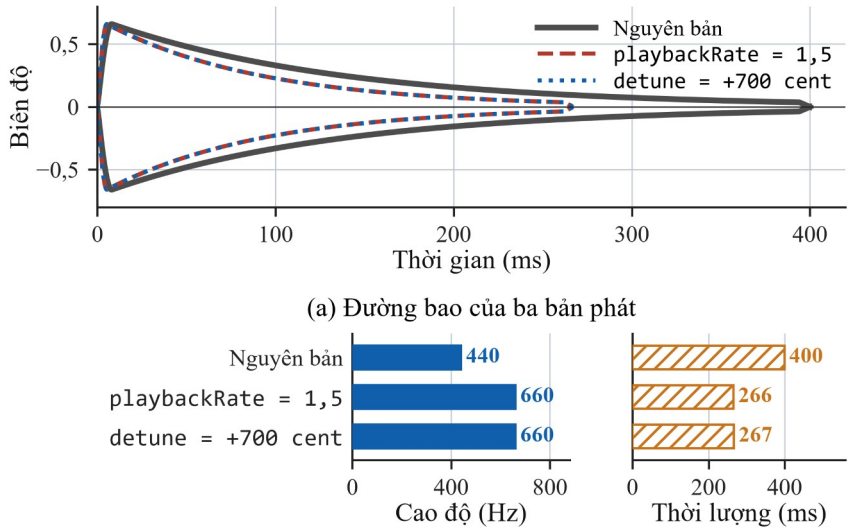
Hậu quả nghe được: chơi một giai điệu bằng răng cưa tự tính thì những nốt cao nghe "lạc giọng", và càng lên cao càng lạc — vì các vạch gập ngược đi *xuống* trong khi nốt đi *lên*.

Đây là lý do nên dùng OscillatorNode thay vì tự sinh bảng sóng, trừ khi cố ý muốn tiếng nhiễu kiểu máy cũ.

11.4.3 playbackRate và detune — hai núm, một đại lượng

$$r = \text{playbackRate} \times 2^{d/1200}\,; \qquad T_{\text{ra}} = T_{\text{gốc}}/r \tag{11.3}$$

Cả hai núm tác động vào cùng một tỉ lệ tốc độ đọc, chỉ khác đơn vị: playbackRate tính theo tỉ lệ thẳng, còn detune tính theo cent của mục 2.6. Chúng *nhân* với nhau chứ không cộng.



(a) Đường bao của ba bản phát

(b) Cao độ đo được và thời lượng đo được

Hình 11.12 Cùng một bộ đệm, ba cách phát, và hai đại lượng đo được

Bảng 11.8 Đo lại trên ba bản phát cùng một bộ đệm

Cách phát	Tỉ lệ r	Cao độ đo được	Thời lượng đo được
Nguyên bản	1,000	440 Hz	400 ms
playbackRate = 1,5	1,500	660 Hz	266 ms
detune = +700 cent	1,498	660 Hz	267 ms

Hai dòng cuối trùng nhau vì bảy trăm cent — một quãng năm đúng — cho tỉ lệ 1,498, gần bằng 1,5. Và cả hai đều làm ngắn thời lượng đúng theo tỉ lệ ấy: bốn trăm chia 1,5 bằng 266 mili giây.

Không tách được cao độ khỏi thời lượng

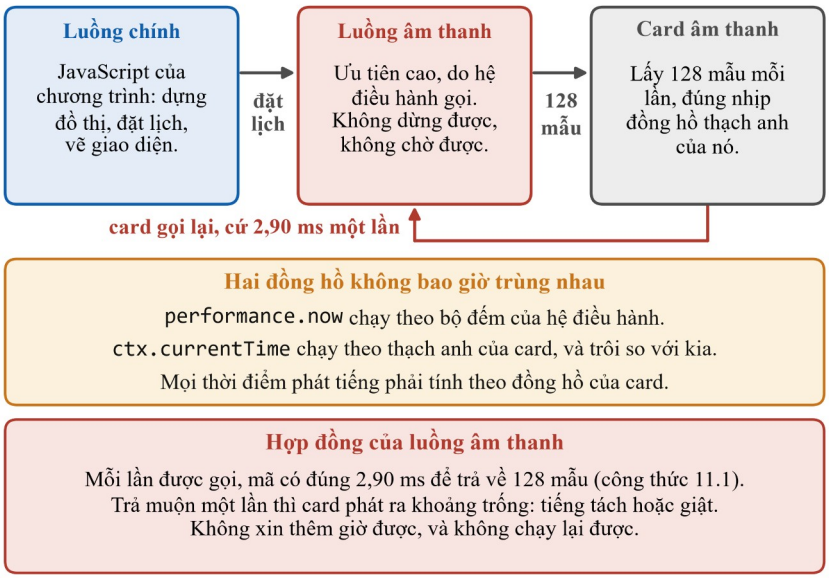
BufferSourceNode chỉ đọc bộ đệm nhanh hay chậm hơn. Đó đúng là phép đổi tần số lấy mẫu của mục 3.7, và nó đổi cả hai thứ cùng lúc.

Muốn đổi cao độ mà *giữ* thời lượng — hoặc ngược lại — thì phải làm việc khác hẳn, dựa trên phân tích và tổng hợp lại theo khung. Web Audio không có sẵn thứ ấy.

Hệ quả thực dụng cho trò chơi: đổi cao độ tiếng bước chân bằng detune là cách rẻ và hay dùng, nhưng chỉ dùng được trong khoảng hẹp, chừng cộng trừ hai trăm cent, trước khi thời lượng lệch tới mức nghe ra.

11.5 Xử lý thời gian thực — trễ và ngân sách

11.5.1 Ai gọi ai



Hình 11.13 Ba bên trong một vòng lặp thời gian thực

Đây là điều phải hiểu trước khi viết bất cứ dòng nào. Card âm thanh chạy theo thạch anh của chính nó và không chờ ai. Cứ mỗi khối, nó đòi lấy đủ số mẫu

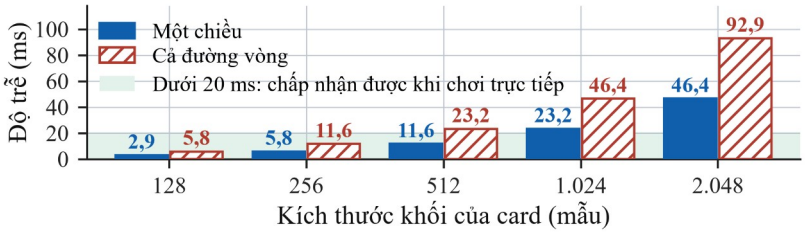
tiếp theo. Trình duyệt gọi luồng âm thanh, luồng âm thanh chạy qua toàn bộ đồ thị node, rồi trả về.

Chương trình JavaScript của chúng ta không nằm trong vòng lặp ấy. Nó ở luồng chính, làm việc khác — vẽ giao diện, đọc phím, đặt lịch — và giao tiếp với luồng âm thanh qua hai đường duy nhất: AudioParam và cổng tin nhắn.

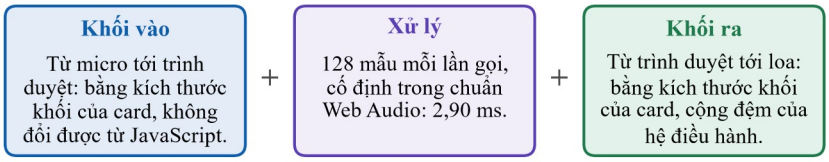
11.5.2 Ba chỗ trễ, và cộng lại là bao nhiêu

$$t_{\text{vòng}} = \frac{L_{\text{vào}} + L_{\text{ra}}}{f_s} + t_{\text{thiếtbị}} \tag{11.4}$$

Trong đó L_vào và L_{ra} là kích thước khối của card ở hai chiều, còn t_thiết bị gồm mọi thứ ngoài tầm với: bộ chuyển đổi, đường truyền không dây, đệm của hệ điều hành. Chú ý khối 128 mẫu của Web Audio *không* xuất hiện trong công thức này — nó nhỏ hơn hẳn và thường bị che khuất bởi hai số kia.



(a) Độ trễ theo kích thước khối



(b) Ba chỗ trễ cộng lại, theo công thức (11.4)

`ctx.baseLatency` và `ctx.outputLatency` báo hai số đầu: nên độ trễ đo được chứ không phải đoán.

Hình 11.14 Độ trễ theo kích thước khối, và ba chỗ trễ

Bảng 11.9 Độ trễ theo kích thước khối của card, ở 44,1 kHz

Khối	Một chiều	Cả đường vòng	Nhận xét
128 mẫu	2,9 ms	5,8 ms	Lý tưởng cho chơi đàn trực tiếp, nhưng tốn CPU nhất
256 mẫu	5,8 ms	11,6 ms	Vẫn thoải mái dưới ngưỡng nhận ra
512 mẫu	11,6 ms	23,2 ms	Bắt đầu nhận ra khi gõ phím đàn; vẫn dùng được cho trò chơi
1.024 mẫu	23,2 ms	46,4 ms	Nghe rõ độ trễ; chỉ hợp cho phát lại, không hợp cho chơi
2.048 mẫu	46,4 ms	92,9 ms	Gần một phần mười giây — không dùng cho tương tác được

Một mốc để nhớ

Dưới 10 mili giây thì gần như không ai nhận ra.

Từ 10 tới 20 mili giây thì người chơi nhạc cụ nhận ra nhưng còn chơi được — đây là vùng mà phần lớn trò chơi và ứng dụng web nhắm tới.

Trên 30 mili giây thì tay và tai lệch nhau rõ, và người chơi bắt đầu đánh sớm lên để bù. Đó là lúc phải hạ kích thước khối, dù tốn CPU hơn.

Trình duyệt báo hai số đầu qua ctx.baseLatency và ctx.outputLatency — nên *đo* được chứ không phải đoán.

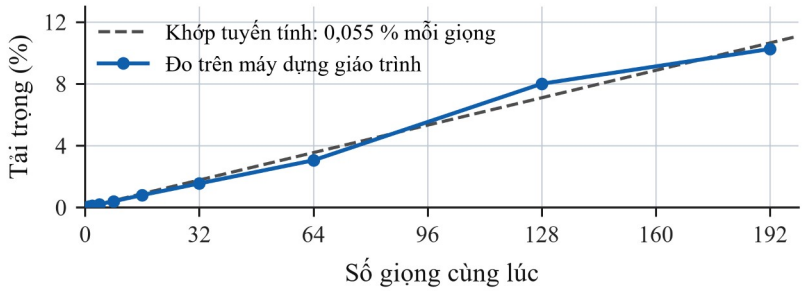
11.5.3 Ngân sách tính toán

Mỗi lần được gọi, mã của chúng ta có đúng một khối thời gian để trả về một khối mẫu. Tỉ số giữa thời gian thật sự dùng và thời gian có được là đại lượng quyết định.

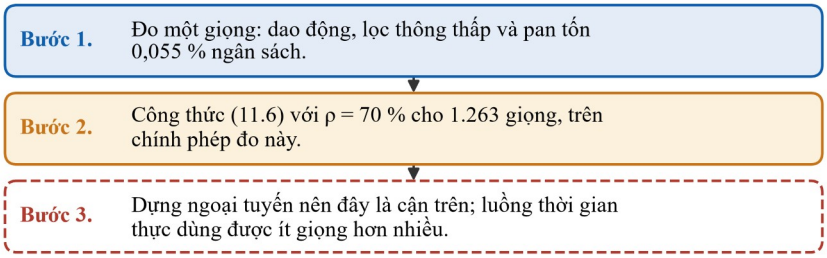
$$\rho = \frac{t_{\text{xử lý}}}{t_q}; \qquad \rho < 1 \tag{11.5}$$

Nếu tỉ số ấy vượt quá 1 thì trả kết quả muộn, và card phát ra khoảng trống. Từ đó suy ra số giọng chạy nổi.

$$N_{\max} = \left\lfloor \frac{\rho_{\max} t_q}{t_{\text{giọng}}} \right\rfloor \tag{11.6}$$



(a) Tải trọng theo số giọng, đo ngoại tuyến



(b) Cách suy ra số giọng chạy nổi

Hình 11.15 Tải trọng theo số giọng, và cách suy ra số giọng chạy nổi

Điều đáng tin ở phép đo trong hình không phải con số cuối mà là *hình dạng* đường cong: nó gần như thẳng qua gốc. Nghĩa là mỗi giọng thêm vào tốn đúng chừng ấy chứ không tốn hơn — nên dự đoán được.

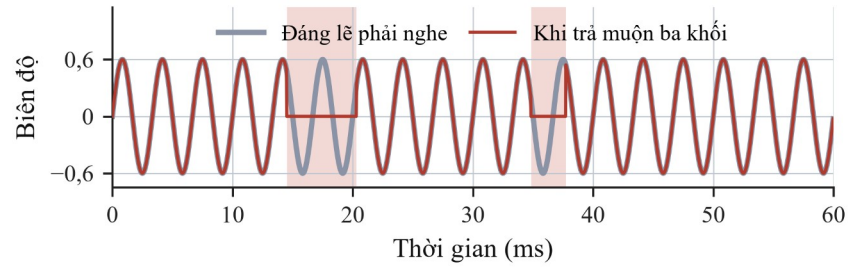
Đọc con số trong hình cho đúng

Phép đo dựng *ngoại tuyến* trên máy dựng cuốn sách này. Dựng ngoại tuyến nhẹ hơn luồng thời gian thực vì nó không có thời hạn cứng và không chia CPU với giao diện.

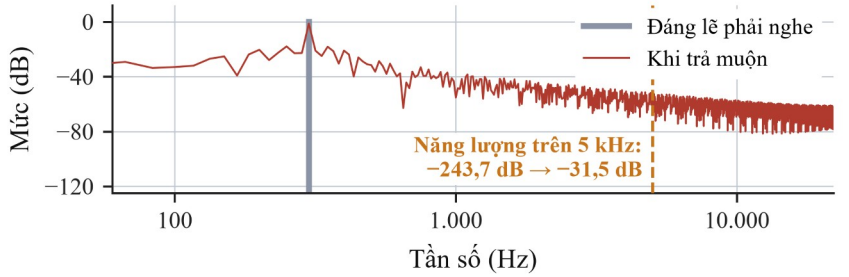
Vì thế con số suy ra là cận *trên*, không phải con số dùng được. Trong một trình duyệt thật, số giọng dùng được thấp hơn nhiều.

Thứ mang đi được là *cách* suy ra: đo một giọng trên máy đích, chia cho ngân sách, rồi chừa ba mươi phần trăm. Đúng cách chừa headroom của mục 10.2, chỉ khác đơn vị.

11.5.4 Vượt ngân sách nghe ra thế nào



(a) Ba khối bị trả muộn trong 60 ms



(b) Phổ: tiếng tách rải năng lượng ra cả dải

Hình 11.16 Ba khối bị trả muộn, và dấu vết của chúng trong phổ

Khi mã trả kết quả muộn, card không chờ: nó phát ra đúng thứ đang có trong bộ đệm, thường là số 0. Mỗi mép của khoảng trống là một bước nhảy đột ngột, đúng thứ đã gây tiếng tách ở mục 6.6.

Bảng 11.10 Đo trên một sóng sin 300 Hz mất ba khối trong 60 ms

Đại lượng	Bình thường	Khi mất ba khối
Phần năng lượng trên 5 kHz	−243,7 dB	−31,5 dB
Nghe ra sao	Một nốt sạch	Một nốt kèm ba tiếng tách

Con số thứ nhất đáng chú ý: một sóng sin sạch gần như không có gì trên 5 kilohertz. Ba khoảng trống trong sáu mươi mili giây đủ để đẩy phần ấy lên hơn hai trăm decibel. Đó là lý do phải chừa dư địa ngân sách chứ không chạy sát trần.

Ba cách xử lý khi tải trọng quá cao

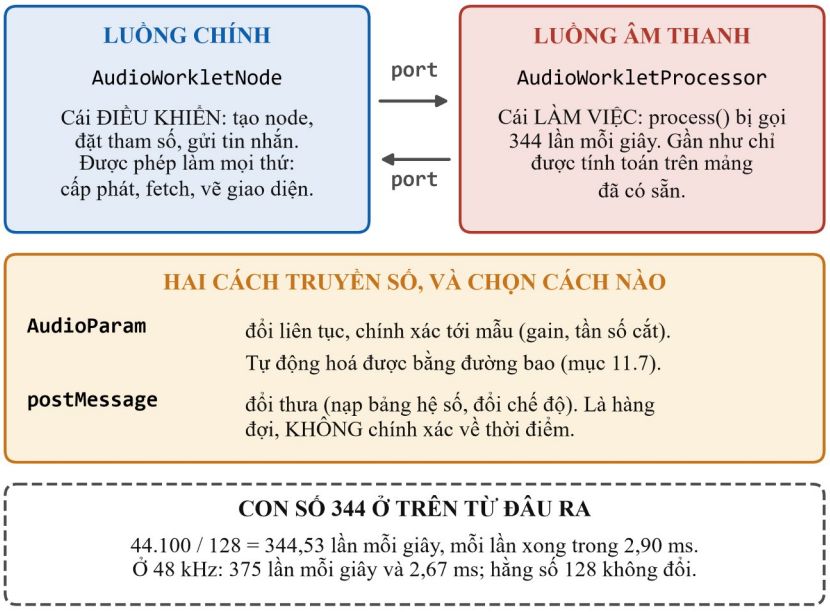
Giảm số giọng: dừng những giọng đã tắt gần hết, hoặc gộp nhiều giọng giống nhau thành một.

Giảm chất lượng: hạ bậc bộ lọc, bỏ bớt tầng vang, đổi từ ConvolverNode sang bộ vang thuật toán của mục 9.7.

Tăng kích thước khối của card: rẻ nhất, nhưng đổi lại độ trễ tăng gấp đôi mỗi lần. Đây là đánh đổi trung tâm của cả phần thời gian thực.

11.6 AudioWorklet — mang mã riêng vào

11.6.1 Hai nửa của một worklet



Hình 11.17 Hai luồng, và hai cách truyền số qua ranh giới

Một AudioWorklet gồm hai nửa sống ở hai luồng khác nhau. Nửa ở luồng chính là AudioWorkletNode — cái điều khiển. Nửa ở luồng âm thanh là AudioWorkletProcessor — cái làm việc.

$$y[n] = f(x[n],\ \mathbf{p}[n]), \qquad n = 0, \ldots, 127 \tag{11.7}$$

Hàm process nhận vào một khối 128 mẫu cùng với giá trị các tham số, và phải ghi đủ 128 mẫu ra. Nó bị gọi 344,53 lần mỗi giây ở 44,1 kilohertz, và mỗi lần có 2,90 mili giây.

Liệt kê 11.4 Khung của một AudioWorkletProcessor

```
// tệp riêng, ví dụ wk-nen.js
class Nen extends AudioWorkletProcessor {
  static get parameterDescriptors() {
    return [
      { name: 'nguong', defaultValue: -18,
        automationRate: 'k-rate' },
      { name: 'tiso', defaultValue: 4, automationRate: 'k-rate' },
    ];
  }
  constructor() { super(); this.z = 0; }   // trạng thái giữa các khối
  process(inputs, outputs, params) {
    const a = inputs[0], b = outputs[0];
    if (!a || !a[0]) { for (const ch of b) ch.fill(0); return true; }
    // ... vòng lặp 128 mẫu ở đây ...
    return true;                            // còn sống thì trả true
  }
}
registerProcessor('nen', Nen);
```

Liệt kê 11.5 Và nửa còn lại, ở luồng chính

```
await ctx.audioWorklet.addModule('wk-nen.js');   // nạp MỘT lần
const w = new AudioWorkletNode(ctx, 'nen');
w.parameters.get('nguong').value = -18;
w.parameters.get('tiso').value = 4;
src.connect(w).connect(ctx.destination);
```

Ba điều dễ sai ở khung trên

Hàm addModule là bất đồng bộ và phải chờ xong *trước* khi tạo node. Quên await là lỗi "processor chưa đăng ký".

Tệp worklet chạy trong một phạm vi riêng: nó không thấy biến toàn cục của trang, không dùng được import thông thường, và không có window hay document.

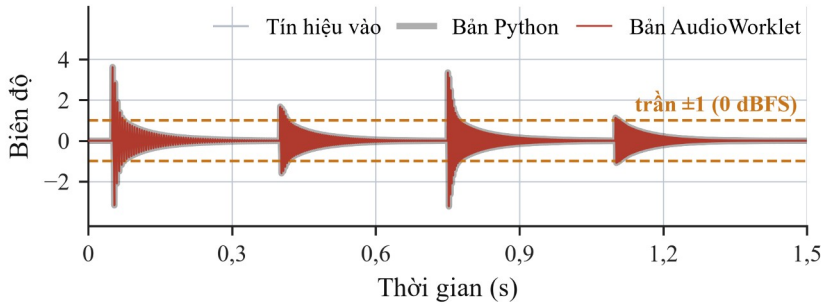
Trả về false là xin bị huỷ — node im vĩnh viễn. Chỉ trả false khi chắc chắn không bao giờ phát ra tiếng nữa.

11.6.2 Kiểm chéo: bộ nén của chương 8 chạy trong trình duyệt

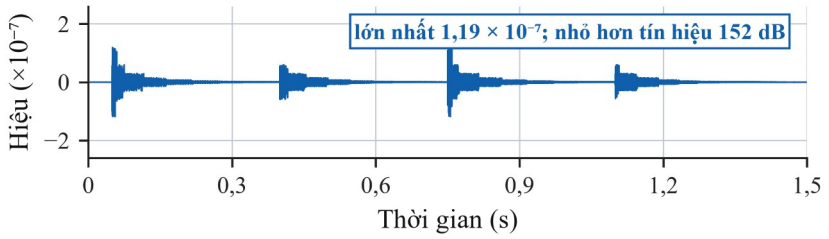
Đây là phép kiểm quan trọng nhất của mục này. Thuật toán bộ nén của mục 8.3 được chép nguyên sang JavaScript — cùng cách dò mức, cùng đường đặc tuyến tĩnh, cùng cách làm trơn trên *gain* chứ không trên mức vào — rồi chạy như một AudioWorkletProcessor.

Liệt kê 11.6 Bộ nén của mục 8.3, viết bằng JavaScript

```
process(i, o, p) {
  const a = i[0], b = o[0];
  const T = p.nguong[0], R = p.tiso[0];
  const at = Math.exp(-1 / (p.tancong_ms[0] / 1000 * sampleRate));
  const re = Math.exp(-1 / (p.nha_ms[0] / 1000 * sampleRate));
  const bu = -(T * (1 - 1 / R));          // gain bù tự động
  for (let n = 0; n < b[0].length; n++) {
    const L = 20 * Math.log10(Math.max(Math.abs(a[0][n]), 1e-9));
    const gr = L > T ? (T + (L - T) / R) - L : 0;
    const k = gr < this.z ? at : re;       // đi xuống là tấn công
    this.z = k * this.z + (1 - k) * gr;
    const g = Math.pow(10, (this.z + bu) / 20);
    for (let c = 0; c < b.length; c++) b[c][n] = a[c][n] * g;
  }
  return true;
}
```



(a) Hai bản cài đặt vẽ chồng lên nhau



(b) Hiệu của hai bản

Hình 11.18 Bản Python và bản AudioWorklet, và hiệu của chúng

Bảng 11.11 Kết quả phép kiểm chéo

Đại lượng	Giá trị
Lệch lớn nhất giữa hai bản	$1{,}19 \times 10^{-7}$
So với mức tín hiệu	Nhỏ hơn 152 decibel
Kết luận	Đúng bằng sai số làm tròn của số thực 32 bit — hai bản là *một*

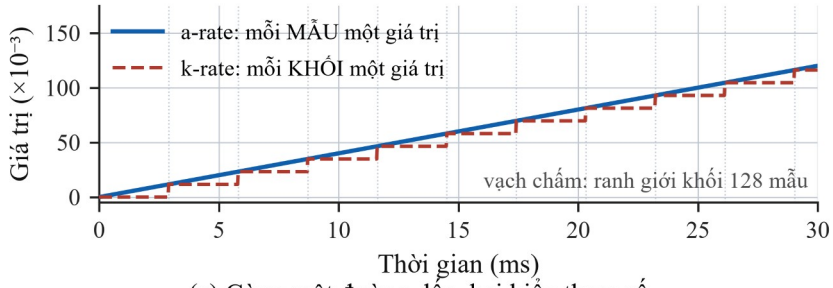
Điều đáng nhớ nhất của cả chương

Mọi thuật toán đã viết bằng Python trong mười chương qua đều chạy được trong trình duyệt, không phải học lại gì về xử lý tín hiệu.

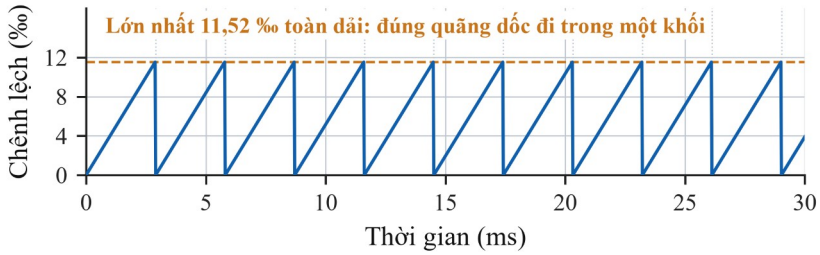
Thứ phải học lại chỉ là cách *đưa* nó vào: chia theo khối 128 mẫu, giữ trạng thái trong đối tượng, và không làm gì bị cấm trong luồng âm thanh.

Chú ý một chi tiết trong hình: gain bù tự động đẩy đỉnh vượt trần 0 dBFS. Web Audio không cắt giúp — đúng như mục 11.2.3 — nên vẫn phải nối bộ giới hạn của mục 8.4 phía sau.

11.6.3 Tham số a-rate và k-rate



(a) Cùng một đường dốc, hai kiểu tham số



(b) Chênh lệch k-rate so với a-rate

Hình 11.19 Cùng một đường dốc, hai kiểu tham số, và chênh lệch giữa chúng

Mỗi AudioParam khai báo trong một worklet phải chọn một trong hai chế độ, và chọn sai thì hoặc tốn vô ích hoặc nghe ra tiếng bậc thang.

Bảng 11.12 Hai chế độ tham số

Chế độ	Trả về gì mỗi khối	Chọn cho	Cái giá
a-rate	Đủ 128 giá trị — một cho mỗi mẫu	Gain, tần số cắt, mọi thứ mà tai nghe ra được sự đứt quãng	Tốn hơn, và mã trong vòng lặp phải đọc mảng
k-rate	Đúng *một* giá trị cho cả khối	Ngưỡng, tỉ số, chế độ — những gì đổi thưa	Chậm sau a-rate đúng một khối, tức 2,90 ms

Đo trên chính đầu ra của một worklet cho thấy chênh lệch lớn nhất giữa hai chế độ là 11,52 phần nghìn của toàn dải — đúng bằng quãng mà đường dốc đi được trong một khối 128 mẫu. Với tham số đổi thưa thì không ai nghe ra khác biệt ấy; với gain thì nghe ra ngay.

Liệt kê 11.7 Đọc tham số cho đúng cả hai chế độ

```
const h = params.hesoGain;          // có thể dài 1 hoặc 128
for (let n = 0; n < b[0].length; n++) {
  const g = h.length > 1 ? h[n] : h[0];    // MẪU CHUẨN, luôn viết thế này
  b[0][n] = a[0][n] * g;
}
```

Dòng thứ ba là mẫu chuẩn phải thuộc lòng. Web Audio được phép trả về mảng dài 1 ngay cả với tham số a-rate, khi giá trị không đổi trong cả khối — đó là một phép tối ưu. Mã không kiểm độ dài sẽ chạy sai một cách im lặng.

11.6.4 Những thứ không được làm trong process

ĐỪNG LÀM	VÌ SAO	LÀM THAY BẰNG
Cấp phát mảng mới	Có thể kích bộ dọn rác, và nó dừng luồng	Cấp sẵn trong constructor, tái dùng
`console.log`	Mỗi dòng là một lần vượt ranh giới luồng	Gom số liệu, gửi về bằng port
Vòng lặp không chặn trên	Một khối quá 2,90 ms là ĐỨT TIẾNG	Chia việc ra nhiều khối, giữ trạng thái
`await, Promise, fetch`	Không chờ được; hàm phải trả về ngay	Nạp trước ở luồng chính, gửi vào bằng port
Đọc ngày giờ của hệ thống	Đó không phải đồng hồ âm thanh (mục 11.3)	Dùng biến currentTime trong worklet
Trả về false quá sớm	Trả false là xin bị huỷ; node im vĩnh viễn	Trả true chừng nào còn phát ra tiếng
Chia cho một số có thể bằng 0	NaN lan ra cả đồ thị rồi im hẳn	Kẹp mẫu số, như 1e-9 ở mục 7.2

Sáu dòng đầu là vấn đề hiệu năng; dòng cuối là vấn đề đúng sai, và là lỗi khó tìm nhất trong bảy dòng.

Hình 11.20 Bảy điều cấm trong luồng âm thanh

Sáu dòng đầu của bảng là chuyện hiệu năng: vi phạm thì thỉnh thoảng đứt tiếng, và lỗi ấy khó tái hiện vì nó phụ thuộc máy và phụ thuộc lúc. Dòng cuối là chuyện đúng sai, và nó là lỗi khó tìm nhất trong bảy dòng.

Vì sao một số NaN là lỗi khó tìm nhất

Chỉ cần một phép chia cho 0 hoặc một loga của số âm là ra NaN.

NaN nhân với bất cứ gì cũng ra NaN, nên nó lan qua mọi node phía sau trong đúng một khối. Toàn bộ đồ thị im hẳn.

Không có thông báo lỗi nào, không có ngoại lệ nào. Chỉ có im lặng.

Cách phòng: kẹp mọi mẫu số và mọi đối số của loga, đúng như hằng số 1e-9 đã dùng suốt từ mục 7.2. Cách tìm: dựng lại đồ thị bằng OfflineAudioContext rồi soát mảng kết quả xem NaN xuất hiện ở mẫu thứ mấy.

11.7 Lập lịch sự kiện và đường bao tham số

11.7.1 Mọi AudioParam đều là một đường theo thời gian

Đặt thẳng giá trị bằng cách gán vào thuộc tính value là cách dùng thô sơ nhất của một AudioParam, và cũng là cách gây tiếng tách. Cách đúng là mô tả cả một *đường*: giá trị này, tại thời điểm kia, đi tới đó theo kiểu nào.

Có ba kiểu đường, và ba kiểu ấy đều đã gặp ở các chương trước dưới tên khác.

$$v(t) = v_0 + (v_1 - v_0)\frac{t - t_0}{t_1 - t_0} \tag{11.8}$$

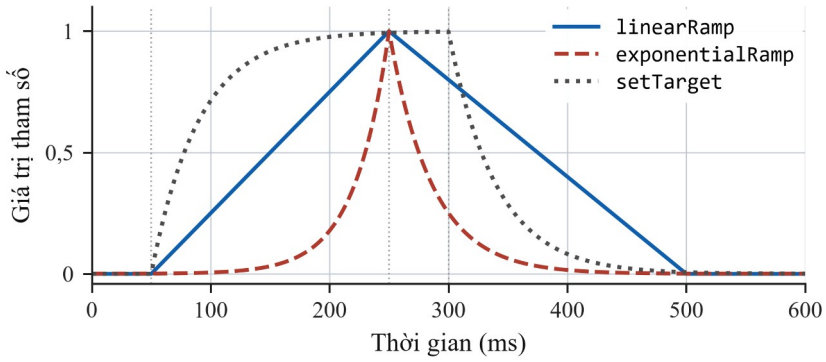
Đường thẳng: đi đều từ điểm này tới điểm kia. Đây đúng là fade tuyến tính của mục 6.6.

$$v(t) = v_0\left(\frac{v_1}{v_0}\right)^u, \qquad u = \frac{t - t_0}{t_1 - t_0}, \qquad v_0 v_1 \neq 0 \tag{11.9}$$

Đường mũ: đi theo *tỉ lệ* chứ không theo hiệu, nên nghe đều trong tai — đúng cách tai cảm nhận độ to ở mục 1.5. Điều kiện cuối công thức là bắt buộc: cả hai đầu đều phải khác 0.

$$v(t) = v_1 + (v_0 - v_1)\, e^{-(t - t_0)/\tau} \tag{11.10}$$

Đường tiến dần: không có đích cuối, chỉ có một hằng số thời gian. Đây chính là bộ làm trơn một cực của mục 8.2, nay hiện ra dưới dạng một lời gọi hàm.



(a) Ba kiểu đường bao tính theo công thức (11.8) đến (11.10)

`linearRamp` (11.8): đi thẳng; dùng cho pan và đại lượng tuyến tính.

`exponentialRamp` (11.9): đi theo tỉ lệ nên nghe đều; không qua 0.

`setTarget` (11.10): tiến dần về đích; sau 3τ mới đạt 95 %.

(b) Cách chọn kiểu đường bao

Hình 11.21 Ba kiểu đường bao đo trên đầu ra thật, và cách chọn

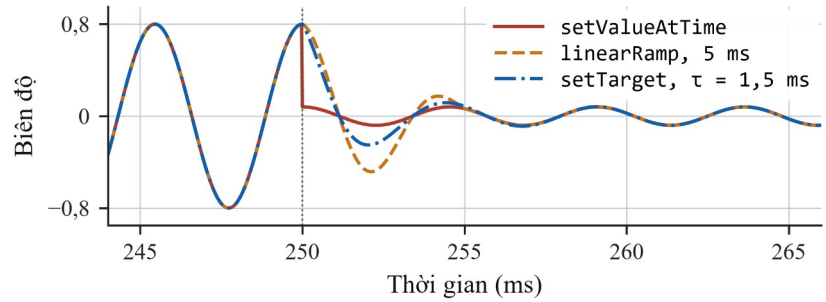
Một điểm dễ sai của đường tiến dần

setTargetAtTime không bao giờ *chạm* tới đích. Sau một hằng số thời gian nó đi được 63 phần trăm quãng đường, sau ba hằng số thì 95 phần trăm.

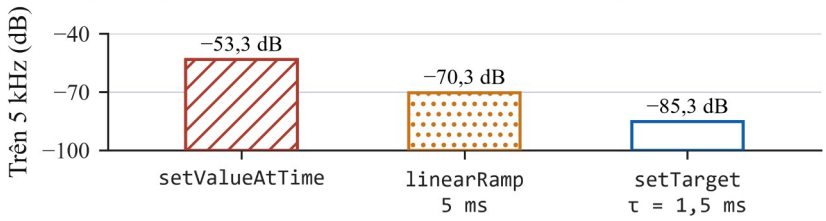
Hệ quả: gọi setTargetAtTime để tắt hẳn một tiếng thì tiếng ấy không bao giờ tắt hẳn, và node nguồn cũng không bao giờ tự dừng.

Cách dùng đúng: dùng nó cho những chỗ chỉ cần "về gần", rồi đặt thẳng về 0 sau ba tới năm hằng số thời gian.

11.7.2 Đổi tham số thế nào cho không kêu tách



(a) Hạ gain từ 0,8 xuống 0,08 tại mốc 250 ms (sóng sin 220 Hz)



(b) Năng lượng trên 5 kHz: càng thấp càng ít tiếng tách

Hình 11.22 Hạ gain bằng ba cách khác nhau, và tiếng tách đo được

Đây là mục 6.6 quay lại, lần này bằng ba lời gọi hàm khác nhau trên cùng một AudioParam. Phép đo dùng đúng đại lượng của mục ấy: phần năng lượng nằm trên 5 kilohertz.

Bảng 11.13 Ba cách hạ gain từ 0,8 xuống 0,08 tại mốc 250 ms

Cách gọi	Năng lượng trên 5 kHz	Nhận xét
setValueAtTime — đặt thẳng	−53,3 dB	Một bước nhảy đúng ở mẫu ấy. Đây là tiếng tách
linearRampToValueAtTime, 5 ms	−70,3 dB	Tốt hơn 17 decibel. Nhưng vẫn còn hai chỗ *gãy*: đầu và cuối đường dốc
setTargetAtTime, τ = 1,5 ms	−85,3 dB	Tốt hơn 32 decibel dù thời gian chuyển *ngắn* hơn — vì đường mũ không có chỗ gãy nào cả

Kết quả này ngược với trực giác

Trực giác nói: chuyển càng lâu càng êm. Nhưng đường tiến dần với hằng số thời gian chỉ 1,5 mili giây lại êm hơn hẳn đường thẳng 5 mili giây.

Lý do không nằm ở độ dài mà nằm ở *đạo hàm*. Đường thẳng có hai chỗ đạo hàm nhảy bậc; đường mũ thì đạo hàm liên tục ở mọi điểm.

Đây cũng là lý do fade hình cosin của mục 6.6 êm hơn fade tuyến tính cùng độ dài. Cùng một nguyên lý, hai chỗ khác nhau trong cuốn sách.

Liệt kê 11.8 Ba cách gọi, và điểm dễ sai của đường mũ

```
// LUÔN đặt một mốc trước khi vẽ đường, nếu không đường sẽ bắt đầu
// từ sự kiện gần nhất TRƯỚC đó — có thể là tận thời điểm 0
g.gain.setValueAtTime(g.gain.value, t);

g.gain.linearRampToValueAtTime(0.08, t + 0.005);      // cách 1
g.gain.setTargetAtTime(0.08, t, 0.0015);              // cách 2

// Cách 3 — đường mũ — KHÔNG đi về 0 được:
// g.gain.exponentialRampToValueAtTime(0, t + 0.005);  →  LỖI
g.gain.exponentialRampToValueAtTime(0.0001, t + 0.005);
g.gain.setValueAtTime(0, t + 0.006);                  // rồi tắt hẳn
```

Hai dòng chú thích đầu là lỗi hay gặp nhất trong cả mục. Một lời gọi ramp luôn bắt đầu từ sự kiện gần nhất trước nó; nếu chưa có sự kiện nào thì nó bắt đầu từ thời điểm 0. Quên đặt mốc thì thay vì một đường dốc 5 mili giây, chúng ta được một đường dốc kéo dài từ lúc trang mở tới giờ.

11.7.3 Lịch theo nhịp

$$t_k = t_0 + k\frac{60}{\text{BPM}\cdot d} \tag{11.11}$$

Trong đó BPM là số nhịp mỗi phút và d là số nốt trong một nhịp: d bằng 1 là nốt đen, 2 là móc đơn, 4 là móc kép. Ở 120 nhịp một phút, nốt móc kép cách nhau 125 mili giây.

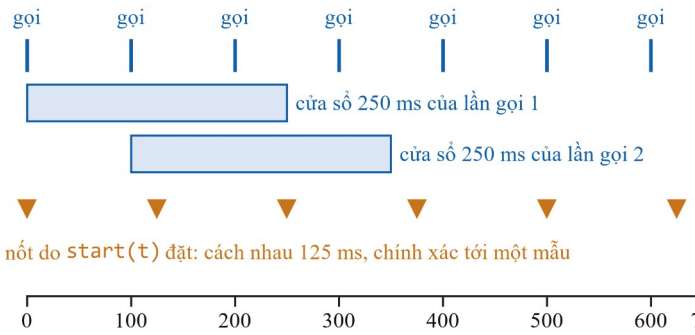
Điều quan trọng là mọi t_k phải tính theo ctx.currentTime, không bao giờ theo đồng hồ của JavaScript. Đã đo ở mục 11.3.2: start(t) chính xác tới một mẫu, còn hẹn giờ của JavaScript lệch tới 8,33 mili giây.

11.7.4 Bộ lập lịch cửa sổ nhìn trước

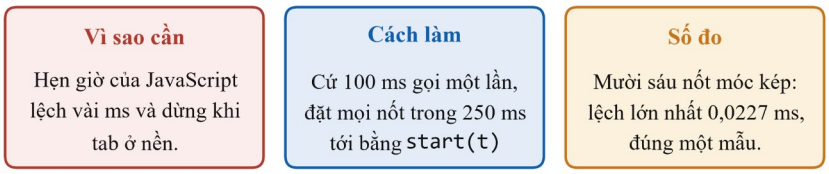
Từ đây ra một mâu thuẫn phải giải. Muốn nhạc chạy vô hạn thì không đặt lịch cho cả bản một lúc được. Nhưng thứ duy nhất đánh thức được chương trình theo chu kỳ lại là hẹn giờ của JavaScript, vốn không đủ chính xác.

Lời giải là mẫu thiết kế trung tâm của mọi ứng dụng nhạc trên web: để hẹn giờ làm việc *thô*, còn việc đặt đúng thời điểm thì giao hẳn cho đồng hồ âm thanh.

$$T_{\text{nhìn}} > T_{\text{gọi}} + J_{\max} \tag{11.12}$$



(a) Hẹn giờ gọi mỗi 100 ms; mỗi lần đặt lịch các nốt trong 250 ms tới



(b) Điều kiện (11.12): cửa sổ > chu kỳ gọi + độ trễ lớn nhất

Hình 11.23 Bộ lập lịch cửa sổ nhìn trước và kết quả đo được

Liệt kê 11.9 Bộ lập lịch cửa sổ nhìn trước, đủ dùng cho một trình tự nhạc

```
let tKe = 0;                    // thời điểm nốt kế tiếp
let stt = 0;                    // số thứ tự nốt
const NHIN = 0.25;              // cửa sổ nhìn trước, giây
const GOI = 100;                // chu kỳ gọi, mili giây

function datLich() {
  while (tKe < ctx.currentTime + NHIN) {
    danhTrong(tKe, stt % 4 === 0);      // tạo node mới, start(tKe)
    tKe += 60 / bpm / 4;                // công thức (11.11)
    stt += 1;
  }
}
setInterval(datLich, GOI);
```

Chọn hai con số này thế nào

Cửa sổ nhìn trước phải rộng hơn chu kỳ gọi *cộng* với độ trễ lớn nhất của hẹn giờ — đó là điều kiện (11.12). Hai trăm tới ba trăm mili giây dùng được cho gần như mọi trường hợp.

Rộng quá thì đổi nhịp hay đổi tiếng chậm phản ứng, vì những nốt đã đặt lịch không rút lại được dễ dàng.

Hẹp quá thì thỉnh thoảng sót một nốt — và lỗi ấy chỉ xuất hiện khi máy bận, nên rất khó tái hiện.

Chu kỳ gọi thì cứ lấy chừng một phần ba tới một nửa cửa sổ. Gọi thừa không tốn gì vì vòng while sẽ không làm gì cả.

Kết quả đo trên chính cách làm này: mười sáu nốt móc kép ở 120 nhịp một phút lệch nhiều nhất 0,0227 mili giây so với thời điểm đặt — đúng một mẫu. Cùng mười sáu nốt ấy đặt bằng hẹn giờ của JavaScript lệch tới 8,33 mili giây, tức hơn ba trăm lần.

Bảng 11.14 Hai cách đặt lịch, đo trên cùng mười sáu nốt

Cách đặt lịch	Lệch lớn nhất	Lệch chuẩn	Nghe ra sao
Cửa sổ nhìn trước, đặt bằng start(t)	0,0227 ms	0,0057 ms	Đều tuyệt đối; lệch nhỏ hơn ngưỡng nghe ra chừng một trăm lần
Đặt thẳng bằng hẹn giờ của JavaScript	8,33 ms	3,88 ms	Nghe rõ là không đều, vì lệch vượt hẳn ngưỡng chừng 2 ms

Hai lỗi hay gặp khi cài bộ lập lịch này

Dùng dấu nhỏ hơn thay vì nhỏ hơn hoặc bằng trong điều kiện vòng while, rồi bỏ sót việc cộng thêm cửa sổ. Kết quả: vòng lặp không bao giờ chạy, và không có tiếng nào cả.

Tạo node ở *ngoài* vòng while rồi gọi start nhiều lần trong đó. Đây là lỗi vòng đời của mục 11.3.3, và nó chỉ lộ ra ở nốt thứ hai.

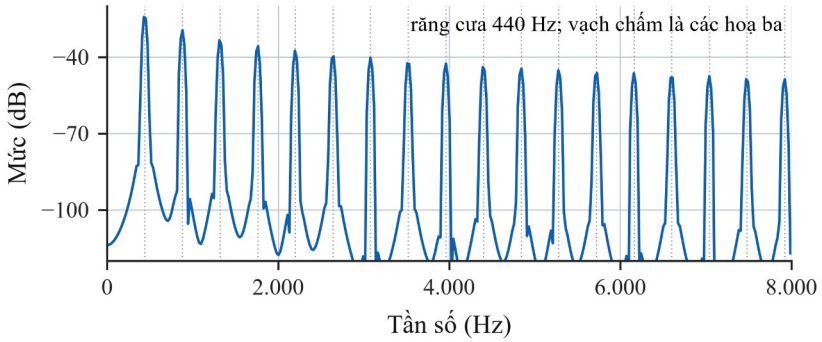
Cả hai đều bắt được bằng cách dựng lại đúng bộ lập lịch ấy trong một OfflineAudioContext rồi đếm số onset trong tệp kết quả.

11.8 Đo và gỡ lỗi trong trình duyệt

11.8.1 AnalyserNode — mắt nhìn vào đồ thị

$$\Delta f = \frac{f_s}{N_{\text{fft}}}; \qquad T_{\text{cửasổ}} = \frac{N_{\text{fft}}}{f_s} \tag{11.13}$$

Hai công thức này là mục 4.6 nguyên vẹn, chỉ đổi tên biến. Đặt fftSize lớn thì mịn về tần số nhưng thô về thời gian, và ngược lại — đánh đổi ấy không có cách nào đi vòng.



(a) Phổ lấy từ `getFloatFrequencyData`

`fftSize` = 2048: 1.024 vạch, cách nhau 21,5 Hz, cửa sổ 46,4 ms (11.13).

`smoothingTimeConstant` = 0,8: trung bình trượt; chỉ đổi cái nhìn thấy.

`minDecibels`, `maxDecibels` = −100, −30 dB: chỉ dùng cho dữ liệu dạng byte.

(b) Ba thuộc tính cấu hình

Hình 11.24 Phổ lấy thẳng từ AnalyserNode, và ba con số phải đặt đúng

Bảng 11.15 Ba con số cấu hình của AnalyserNode

Thuộc tính	Ý nghĩa	Đặt thế nào
fftSize	Số mẫu mỗi khung phân tích; phải là luỹ thừa của 2, từ 32 tới 32768	Đặt 2048 thì được 1024 vạch, mỗi vạch cách 21,5 Hz, cửa sổ dài 46,4 ms
smoothingTimeConstant	Trung bình trượt giữa các khung, từ 0 tới 1	Mặc định 0,8. Nó chỉ đổi cái *nhìn thấy*, không đổi tiếng ra
minDecibels và maxDecibels	Khoảng quy đổi khi lấy dữ liệu dạng byte	Mặc định −100 tới −30 dB. Muốn số thật thì gọi getFloatFrequencyData

Điều quan trọng nhất phải nhớ về node này

AnalyserNode *không* nằm trên đường tín hiệu. Cắm nó vào giữa một chuỗi hay không cắm thì tiếng ra vẫn thế.

Nó cũng không cần nối tới destination mới chạy — chỉ cần có tín hiệu vào là nó phân tích.

Vì thế nó là công cụ gỡ lỗi an toàn: cắm thêm bao nhiêu cái cũng không đổi kết quả, chỉ tốn chút CPU.

11.8.2 Bảy lỗi hay gặp

Triệu chứng	Nguyên nhân thường gặp	Cách kiểm
Không có tiếng gì cả	Ngữ cảnh còn ở trạng thái suspended	In `ctx.state`; gọi `resume()` khi bấm
Nghe được một lần rồi thôi	Gọi `start()` lần thứ hai trên cùng một node	Mục 11.3.3: tạo node mới cho mỗi tiếng
Tiếng tách ở mỗi lần đổi	Đặt thẳng gain bằng `setValueAtTime`	Đo năng lượng cao tần (hình 11.22)
Nhịp không đều	Đặt lịch bằng `setTimeout` thay vì `start(t)`	Đo thời điểm onset (hình 11.23)
Càng chạy càng chậm	Node cũ chưa được thu hồi vì còn tham chiếu	Đếm node sống; gỡ sau `onended`
Méo tiếng khi thêm giọng	Cộng nhiều nhánh mà không chừa headroom	Đo đỉnh; áp công thức đồng pha (mục 10.2.3)
Im hẳn giữa chừng	Một mẫu thành NaN và lan ra cả đồ thị	Kẹp mẫu số; dựng ngoại tuyến rồi soát

Hình 11.25 Bảy lỗi hay gặp và cách kiểm từng lỗi

Cột phải của bảng đáng chú ý: sáu trong bảy cách kiểm là *đo bằng số*, và cả sáu đều dùng công cụ đã viết ở các chương trước. Đó là lợi ích thực dụng của việc tự cài đặt lại mọi thứ trong mười chương qua.

> **Một quy trình gỡ lỗi đáng thuộc**
>
> Bước 1 — dựng lại đúng đồ thị ấy bằng OfflineAudioContext. Nếu lỗi biến mất thì đó là lỗi thời gian thực: ngân sách, hoặc lịch.
>
> Bước 2 — nếu lỗi còn, ghi AudioBuffer ra tệp rồi soát bằng số: tìm NaN, tìm mẫu vượt trần, đo phổ, đo thời điểm onset.
>
> Bước 3 — so với bản Python cùng thuật toán, đúng như phép kiểm chéo ở mục 11.6.2. Hai bản lệch nhau ở đâu thì lỗi ở đó.
>
> Ba bước này biến gỡ lỗi âm thanh từ chuyện nghe ngóng thành chuyện đo đạc — và đó là toàn bộ tinh thần của cuốn sách.

11.9 Thực hành, bài tập và tổng kết

11.9.1 Ba bài thực hành

Ba bài dưới đây làm ngay sau khi đọc xong chương, mỗi bài chừng mười phút. Chỉ cần một trình duyệt và một tệp HTML — không cần cài gì.

- **Bài 1 — đồ thị đầu tiên.** Dựng chuỗi bốn node của mục 11.2.1 với một tệp WAV của người đọc. Rồi đổi ba tham số của bộ lọc và *nghe*, đối chiếu với đáp ứng đã học ở mục 9.3.
- **Bài 2 — đo độ trễ và ngân sách.** In ra ctx.baseLatency và ctx.outputLatency của máy người đọc. Rồi thêm dần số giọng cho tới khi nghe thấy đứt tiếng, và ghi lại con số ấy. So với công thức (11.6).
- **Bài 3 — bộ lập lịch.** Cài bộ lập lịch cửa sổ nhìn trước ở mục 11.7.4 để đánh một nhịp đều. Rồi thay nó bằng setTimeout đặt lịch trực tiếp, và *nghe* sự khác

nhau. Cuối cùng chuyển tab sang cửa sổ khác và nghe cách thứ hai bị lỗi như thế nào.

11.9.2 Bảy bài tập

Bảy bài dưới đây. Bài nào có đáp số thì đáp số in ngay sau đề — điểm nằm ở *cách làm* và ở phần giải thích, không nằm ở con số.

Bảng 11.16 Bảy bài tập của chương 11

#	Đề bài	Đáp số
1	Ở 48 kHz, một khối 128 mẫu kéo dài bao lâu, và process() bị gọi bao nhiêu lần mỗi giây?	2,667 ms; 375 lần
2	Card đặt khối 256 mẫu ở 48 kHz. Độ trễ cả đường vòng là bao nhiêu, nếu bỏ qua phần thiết bị?	10,67 ms
3	Một worklet đo được tốn 0,9 ms cho mỗi khối ở 44,1 kHz. Tỉ số ρ bằng bao nhiêu? Giữ tải dưới 70 phần trăm thì chạy nổi mấy bản?	ρ = 0,310, tức 31,0 %; chạy nổi 2 bản
4	Đặt playbackRate = 2 và detune = −1200 cent. Tỉ lệ r bằng bao nhiêu? Một mẫu dài 3 giây phát ra dài bao lâu?	r = 1,000; vẫn 3 giây
5	linearRampToValueAtTime đi từ 0,2 lên 0,9 trong 40 ms. Giá trị tại mốc 25 ms là bao nhiêu?	0,6375
6	setTargetAtTime đi từ 1,0 về 0,1 với τ = 8 ms. Sau bao lâu thì còn cách đích 5 phần trăm quãng đường?	23,97 ms
7	BPM = 96, nốt móc kép. Hai nốt cách nhau bao lâu? Một cửa sổ nhìn trước 200 ms chứa được mấy nốt?	156,25 ms; chứa 2 nốt

11.9.3 Mở rộng cho ai muốn đi xa hơn

- **Bảng sóng tự đặt.** Dùng PeriodicWave để dựng một dạng sóng từ biên độ các hoạ ba do người đọc chọn, rồi so phổ đo được với thiết kế. Đây là cách sinh tiếng đàn ống hoặc tiếng kèn mà vẫn băng giới hạn.
- **Bộ giới hạn thật bằng worklet.** Cài bộ giới hạn có lookahead của mục 8.4 thành một AudioWorklet, rồi kiểm xem nó có giữ được trần khi đặt sau bộ nén ở mục 11.6.2 không.

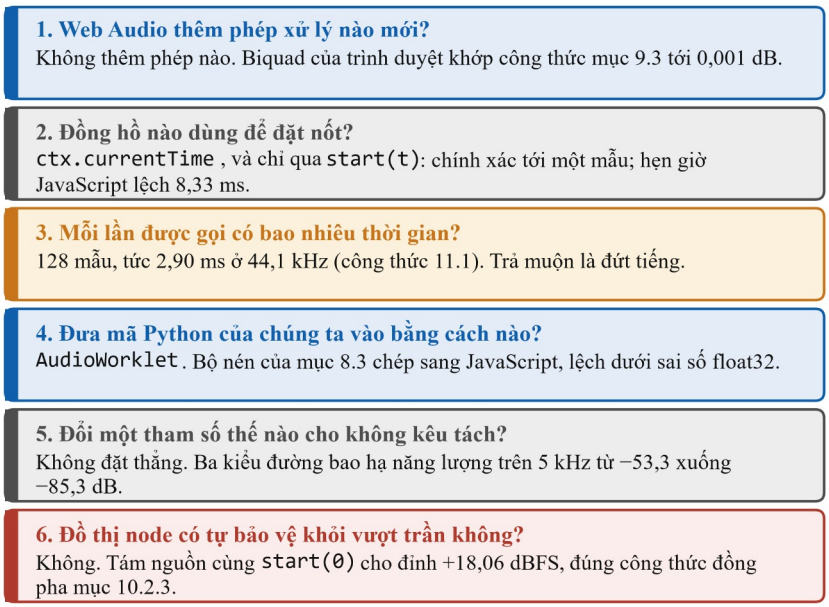
– **Đo tải trọng ngay trong trình duyệt.** Dùng ctx.renderCapacity để đọc tải trọng thật của luồng âm thanh, rồi vẽ nó theo thời gian. So đường cong ấy với đường ở hình 11.20.
– **Bộ đo tương quan hai kênh.** Dùng hai AnalyserNode ở chế độ miền thời gian để vẽ đại lượng của mục 10.2.2 theo thời gian thực. Đây là công cụ bắt lỗi đảo cực đã nói tới ở chương 10.

11.9.4 Câu hỏi tự trả lời

Bảy câu dưới đây không có đáp án in trong sách. Trả lời được cả bảy nghĩa là đã nắm chương này.

– Vì sao ctx.currentTime không dùng được để đặt nốt vào đúng nhịp, trong khi chính nó lại là đồng hồ duy nhất được phép dùng?
– Tám dao động ở tám tần số khác nhau, hệ số tương quan giữa chúng gần 0. Vì sao đỉnh của tổng vẫn khớp công thức *đồng pha*?
– Một node nguồn chỉ start được một lần. Nêu hai lý do kỹ thuật cho thiết kế ấy.
– Vì sao setTargetAtTime với $\tau = 1{,}5$ ms lại êm hơn linearRamp 5 ms, dù nó chuyển nhanh hơn?
– Tăng kích thước khối của card làm giảm nguy cơ đứt tiếng nhưng tăng độ trễ. Giải thích cả hai vế bằng cùng một công thức.
– Trong process(), vì sao phải viết h.length > 1 ? h[n] : h[0] chứ không viết thẳng h[n]?
– Một đồ thị đang chạy bỗng im hẳn, không báo lỗi gì. Nêu ba giả thuyết và cách phân biệt chúng bằng số.

11.9.5 Tổng kết — chương 11 để lại gì



Hình 11.26 Sáu câu phải trả lời được sau chương 11

Bảng 11.17 Những thứ mới của chương 11 và chỗ dùng lại

Thứ mới	Làm gì	Dùng lại ở
AudioContext và OfflineAudioContext	Hai kiểu chạy: thời gian thực và dựng ngoại tuyến	Chương 12 và mọi bài thực hành còn lại
connect và đồ thị node	Nối khối thay vì viết vòng lặp	Chương 12 — rack hiệu ứng
ctx.currentTime và start(t)	Đặt sự kiện chính xác tới mẫu	Chương 12 — trộn động theo sự kiện
Ba kiểu đường bao tham số	Đổi tham số mà không kêu tách	Chương 12 — lập lịch tham số
AudioWorklet	Mang thuật toán riêng vào luồng âm thanh	Chương 12 và chương 14
Bộ lập lịch cửa sổ nhìn trước	Chạy một trình tự nhạc vô hạn mà vẫn đúng nhịp	Chương 12
Ngân sách tính và cách đo nó	Biết trước chạy nổi bao nhiêu giọng	Chương 12 — tối ưu tài nguyên

Một câu cho cả chương

Chương này không thêm một phép xử lý nào. Mọi khối trong trình duyệt đều là thứ đã tự viết, và mục 11.2.4 chứng minh điều ấy bằng ba nguồn số khớp nhau tới 0,001 decibel.

Cái mới nằm ở chỗ khác: quyền điều khiển đổi chủ. Trình duyệt gọi mã của chúng ta cứ 2,90 mili giây một lần, và mọi quyết định thiết kế trong chương này đều là hệ quả của một câu hỏi duy nhất — làm sao trả kết quả đúng hạn.

Chương 12 giữ nguyên khung ấy và hỏi câu tiếp theo: làm sao tổ chức một ứng dụng thật trên nền nó.

Tài liệu tham khảo của chương

Nguồn trực tiếp của nội dung trong bài

- W3C — Web Audio API, khuyến nghị chính thức. Đây là nguồn *trực tiếp* của toàn bộ chương này: hằng số 128 mẫu, danh sách node, ngữ nghĩa của từng phương thức lập lịch, và luật về a-rate với k-rate. Tài liệu dài nhưng tra cứu rất tốt.
- Wilson, C. — A Tale of Two Clocks: Scheduling Web Audio with Precision. Bài viết gốc của mẫu thiết kế cửa sổ nhìn trước ở mục 11.7.4, và là bài nên đọc nguyên văn.
- Choi, H. — Enter AudioWorklet, và loạt bài về kiến trúc luồng của Web Audio. Nguồn của mục 11.6, gồm cả danh sách những thứ không nên làm trong luồng âm thanh.

– Bristow-Johnson, R. — Cookbook formulae for audio EQ biquad filter coefficients. Đây là công thức mà BiquadFilterNode của mọi trình duyệt cài đặt, và là thứ đã dùng ở mục 9.3.

Sách

– Smus, B. — Web Audio API. Sách mỏng, viết sớm nhưng phần mô hình đồ thị node và phần lập lịch vẫn đúng nguyên.
– Adenot, P., Toy, R. (chủ biên) — bản đặc tả kèm ghi chú thiết kế trên kho mã của W3C. Đọc phần thảo luận để hiểu *vì sao* một số quyết định gây bất tiện lại được giữ.
– Reiss, J. D., McPherson, A. — Audio Effects: Theory, Implementation and Application. Dùng để đối chiếu thuật toán khi chuyển một hiệu ứng từ Python sang worklet.

Công cụ nên biết

– Bảng điều khiển của trình duyệt, thẻ Performance. Nó vẽ được thời gian của từng luồng, và luồng âm thanh hiện ra thành một hàng riêng — đây là cách nhìn thấy chuyện vượt ngân sách ở mục 11.5.4.
– Trang about:tracing của trình duyệt nhân Chromium, hoặc about:debugging của Firefox. Chúng cho xem thẳng số lần khối bị trả muộn.
– Thuộc tính ctx.renderCapacity — cách đọc tải trọng thật của luồng âm thanh ngay trong mã, không cần công cụ ngoài.
– node-web-audio-api — một bản cài đặt Web Audio đầy đủ chạy ngoài trình duyệt. Toàn bộ số liệu trong chương này dựng bằng nó, và nó là cách để đưa phép kiểm đồ thị Web Audio vào một vòng kiểm tự động.

Đọc thêm theo hướng

– **Muốn đi sâu về hiệu năng:** tìm đọc về WebAssembly trong AudioWorklet, và về SIMD. Chúng cho phép chạy những thuật toán tốn tính toán hơn nhiều trong cùng ngân sách — nhưng chỉ đáng làm sau khi đã *đo* và biết chỗ nào chậm.
– **Muốn đi sâu về đồng bộ:** tìm đọc về SharedArrayBuffer và về hàng đợi không khoá. Đó là cách truyền dữ liệu lớn qua ranh giới luồng mà không tốn chi phí sao chép của postMessage.
– **Muốn đi sâu về nhạc cụ:** tìm đọc về tổng hợp trừ và về tổng hợp FM, rồi cài lại chúng bằng đúng những node đã học ở chương này. Cả hai đều dựng được mà không cần worklet.
– **Muốn đi sâu về không gian:** tìm đọc về PannerNode ở chế độ HRTF và về Resonance Audio. Chúng là mục 10.7 đã đóng gói sẵn — nhưng bảng HRTF thì cố định, đúng như giới hạn thứ ba đã nêu ở đó.

Một lưu ý về các con số trong bài

> **Cách đọc các con số**
>
> Các con số về độ trễ, về chu kỳ khối, về quy đổi cent và về khoảng cách nhịp là kết quả tính chính xác từ công thức — chúng đúng với mọi máy.
>
> Các con số về đáp ứng bộ lọc, về độ chính xác của lịch, về tiếng tách, về chồng phổ và về phép kiểm chéo bộ nén là kết quả *đo* trên đồ thị Web Audio *chạy thật*, dựng ngoại tuyến rồi đổ ra tệp và đo lại bằng mô-đun Python của các chương trước.
>
> Riêng con số về số giọng chạy nổi ở mục 11.5.3 là số đo thời gian thật trên máy dựng cuốn sách này, và nó là cận *trên* chứ không phải con số dùng được — lý do nêu ngay tại mục ấy.
>
> Mã JavaScript in trong bài là mã đã chạy, không phải mã giả.