42-Tâches, arguments et objets globaux JavaScript

introduction

L'objet *js* vous permet de créer des fonctions JavaScript qui utilisent le planificateur Max. Ces fonctions peuvent être déclenchées par messages Max vers l'objet *js*. L'intervalle de temps auquel la fonction est appelée, le nombre de fois qu'elle se répète (y compris si elle se répète indéfiniment), et si elle commence à s'exécuter immédiatement ou à moment donné dans le futir peuvent tous être déterminés par votre code.

Dans ce didacticiel, nous allons étudier le fonctionnement de la planification en JavaScript. En cours de route, nous examinerons deux autres caractéristiques importantes de l'implémentation de JavaScript dans Max: les arguments d'objet *js* (qui vous permettent de transmettre des ***arguments*** directement à votre code JavaScript à partir de la boîte d'objets) et les objets ***globaux*** (qui vous permettent de partager des données entre les structures de données internes *js* et Max).

Pour ouvrir le patch du didacticiel, cliquez sur le bouton vert **Ouvrir didacticiel** dans le coin supérieur droit de la fenêtre de documentation.

Planification en JavaScript

Jetez un coup d'œil au patcheur du tutoriel. Vous y verrez quatre objets *js*, qui utilisent tous le même fichier source JavaScript (appelé «globaltask.js»), enregistré sur le disque dans le même dossier que le patch du didacticiel.

Cliquez sur l'objet *button* situé en haut du patch et intitulé «send a bounce». Les quatre objets *js* devraient commencer à générer des nombres qui sont ensuite envoyés en aval vers la sortie de votre synthétiseur MIDI (via les objets *makenote* et *noteout*). Double-cliquez sur l'objet *noteout* et sélectionnez un synthétiseur de sortie valide. Vous devriez alors commencer à écouter les notes. Des objets *button* supplémentaires sont connectés à la sortie gauche des objets *js* pour fournir un retour visuel sur le moment où un nombre est envoyé. De plus, un objet *patcheur* appelé **view** reçoit les messages **bang** de l'objet *button* et crée un retour visuel défilant du rythme généré avec un objet *multislider*.

Notez que le timing spécifique des quatre objets, ainsi que les hauteurs de son qu'ils génèrent, sont différents; ceux-ci sont déterminés par les arguments des objets *js* (nous y reviendrons plus tard).

Le code JavaScript utilisé dans les objets *js* est une simple maquette d'une fonction de timing à décroissance exponentielle (analogue à une balle en caoutchouc tombée sur un sol dur). Les notes sont envoyées à un rythme croissant de manière exponentielle, jusqu'à ce que la vitesse d'envoi dépasse une valeur seuil (cinq millisecondes, dans le cas de notre script). En dépassant ce seuil, la fonction s'arrête et un **bang** est envoyé par la sortie droite de nos objets *js* pour notifier que la fonction de chronométrage a cessé. L'utilisation d'un **bang** «done» est une convention commune aux objets Max pour signifier l'achèvement d'une tâche (c.f. *line*, *uzi*, *coll*, etc.).

Cliquez sur le *toggle* intitulé «repeat» et cliquez à nouveau sur le *button* du haut. Remarquez que le cycle des notes rebondissantes se répète, car les **bang** des objets *js* les font se redéclencher. La différence dans l'accélération temporelle des quatre objets *js* les fait se mettre en phase sur plusieurs itérations du cycle (notez comment cela est affiché dans le *multislider*). Si vous cliquez à nouveau sur le *toggle*, le cycle actuel de rebonds se terminera dans chaque objet puis s'arrêtera. Double-cliquez sur n'importe lequel des objets *js* du patch ; examinons leur code.

Dans les temps

Examinez le code global de notre script:

```
inlets = 1;
outlets = 2;

// define global variables and set defaults
var tsk = new Task(mytask, this); // our main task
var count = 0;
var decay = 1.0;
// defaults for arguments
var dcoeff = -0.0002; // decay coefficient
var note = 60; // note to trigger upon bounce
// process arguments (decay coefficient, note to trigger)
if(jsarguments.length>1) // argument 0 is the name of the js file
{
        dcoeff = jsarguments[1];
}
if(jsarguments.length>2)
{
        note = jsarguments[2];
}

// Global (Max namespace) variables
glob = new Global("bounce");
glob.starttime = 500;
```

La section de code global de notre fichier JavaScript, qui définit les entrées, sorties (nous en avons deux cette fois) et variables habituelles, comporte un certain nombre d'éléments que nous n'avons jamais rencontrées. La première est une variable assignée à un objet spécial appelé **Task**:

```
var tsk = new Task(mytask, this); // our main task
```

Ceci crée un nouvel objet Task en JavaScript, désigné par le nom **tsk**. Lorsque nous appelons des méthodes pour **tsk**, elles concernent la programmation d'une fonction appelée **mytask ()**. L'objet de contrôle de la tâche sera notre objet *js* (appelé **this**). Si nous souhaitons exécuter notre tâche une seule fois, nous écrirons:

```
tsk.execute (); // lance notre tâche une fois
```

Si nous souhaitons que notre tâche soit répétée toutes les 250 millisecondes pendant 20 répétitions, nous écrirons:

```
tsk.interval = 250;
tsk.repeat (20);
```

Si aucun argument n'est donné à la méthode **repeat ()**, la tâche sera programmée indéfiniment, jusqu'à ce que nous l'annulons comme suit:

```
tsk.cancel (); // annule notre tâche
```

Les méthodes **execute ()**, **repeat ()** et **cancel ()** nous offrent toute la souplesse dont nous avons besoin pour planifier des événements répétitifs en JavaScript. En plus de la propriété **interval** de l'objet Task, nous pouvons également savoir si une tâche est en cours d'exécution ou non (**running**), et combien de fois elle a été appelée (**itérations**), par exemple.

N'oubliez pas que toutes les méthodes de l'objet *js* (qu'elles soient déclenchées par des messages Max ou planifiées en interne par des tâches) sont exécutées à faible priorité dans le planificateur Max. Cela signifie que, bien qu'elles s'exécuteront toujours et enverront des données à Max dans le bon ordre, on ne peut pas compter sur elles pour obtenir un timing extrêmement précis si le planificateur est surchargé par d'autres actions.

Une fois que nous avons défini notre tâche *tsk*, nous la déclenchons via la méthode **bang ()** vers notre objet *js*:

```
// bang -- start the task
function bang()
{
	tsk.cancel(); // cancel the bounce, if it's going already
	count = 0; // reset the number of bounces
	decay = 1.0; // reset the initial decay
	tsk.interval = glob.starttime; // set the initial task interval
	tsk.repeat(); // start the bouncing
}
```

Lorsque l'un de nos objets *js* reçoit un **bang**, il annule toute tâche tsk précédemment programmée, réinitialise certaines variables pertinentes pour la fonction de la tâche, définit un intervalle de temps initial pour la tâche, puis le relance.

Lancez un "rebond" dans le patch du didacticiel en cliquant sur le bouton en haut. Cliquez sur la boîte de *message* intitulée **stop**. Les notes devraient cesser. Notre fonction **stop ()** annulera notre tâche précédemment programmée en appelant simplement la méthode **cancel ()** à **tsk**:

```
// stop -- allow the user to stop the bouncing
function stop()
{
	tsk.cancel(); // cancel our task
}
```

La tâche à accomplir

Notre objet Task, une fois mis en mouvement par notre méthode **bang ()**, appelle la fonction définie dans sa déclaration initiale.

Parcourez le code de la fonction mytask ():

```
// mytask -- the scheduled task - output number and reschedule next task
function mytask()
{
	if(arguments.callee.task.interval>5) // keep bouncing
	{
		outlet(0, note); // send a note value
		decay = decay*Math.exp(++count*dcoeff); // increment the decay variable
```

```
		arguments.callee.task.interval=arguments.callee.task.interval*decay; // update
the task interval
	}
	else // bounce interval is too small, so consider it 'floored'
	{
		arguments.callee.task.cancel(); // cancel the task
		outlet(1, bang); // send a bang out the right outlet to signify that we're done bouncing
	}
}
```

Contrairement aux autres fonctions que nous avons utilisées dans nos tutoriels JavaScript, nous n'avons pas l'intention que notre fonction **mytask ()** soit déclenchée par un message Max provenant de l'extérieur de l'objet *js*. Par défaut, toute fonction déclarée dans un objet *js* répondra à un message nommé de manière appropriée dans l'environnement Max. Comme nous ne voulons pas que **mytask ()** soit déclenchée par un message **mytask** de notre patcheur, nous plaçons la ligne de code suivante à la fin de la fonction: mytask.local = 1;

Cette instruction rend **mytask ()** locale dans l'environnement *js* et est inaccessible de l'extérieur.

Notre fonction Task accomplit deux choses: elle envoie un entier à Max (déclenchant une note MIDI) et incrémente son propre intervalle de temps afin que la prochaine exécution de **mytask ()** ait lieu un peu plus tôt. En dehors de la fonction de tâche, nous pouvons modifier notre intervalle de temps en définissant la propriété **interval** à la tâche (par exemple, **tsk**.interval = 250). Les propriétés et les méthodes d'un objet Task peuvent être modifiées dans la fonction task en faisant référence à Task en tant que **callee.task**, par exemple:

```
arguments.callee.task.interval=250 ; // adjust timing of task to 250
arguments.callee.task.cancel(); // have the task cancel itself
```

Nous utilisons cette capacité réflexive pour modifier l'intervalle de temps de notre objet Task à l'intérieur de la fonction Task. Lorsque l'intervalle de temps diminue à une valeur suffisamment faible (5 millisecondes dans notre cas), nous utilisons également cette fonctionnalité pour que notre fonction Task annule la tâche qui l'a appelée en premier lieu.

Utilisation d'un objet mathématique

Examinez à nouveau le code de **mytask ()** en prêtant attention à la ligne qui change la valeur de décroissance à chaque rebond:

```
decay = decay*Math.exp(++count*dcoeff); // increment the decay variable
```

En plus des les objets permettant une interaction entre JavaScript et Max (**Maxobj**, **Task**), JavaScript possède un certain nombre d'objets de base pouvant être utiles lors de l'écriture de programmes pour *js*. L'objet **Math** possède une grande bibliothèque de propriétés et de méthodes intégrées qui vous permettent d'exécuter des fonctions mathématiques les plus courantes. Dans notre code, nous utilisons la méthode **exp ()** avec l'objet Math, qui renvoie la valeur de **e** (la base du logarithme naturel: environ **2,71828**) à la puissance de son argument (dans ce cas, notre coefficient de décroissance multiplié par le nombre de billes suivant). Ceci est crucial pour la modélisation du taux de croissance exponentielle de l'événement de rebond.

Les caractéristiques de l'objet Math en JavaScript est à peu près analogue à celle de la bibliothèque mathématique en C ou à l'objet *expr* de Max (qui est lui-même basé sur la bibliothèque mathématique en C). Un certain nombre d'autres objets de base prédéfinis (par exemple **Date**, **String**) fournissent des extensions similaires au langage qui correspondent plus ou moins à leurs équivalents C (par exemple, time, string).

Arguments de l'objet *js*

Deux des variables de notre script (**dcoeff** et **note**) sont déterminées par les arguments donnés à l'objet *js*. Ces arguments sont analysés dans notre bloc de code global en vérifiant la propriété **jsarguments** de notre objet *js*:

```
// process arguments (decay coefficient, note to trigger)
if(jsarguments.length>1) // argument 0 is the name of the js file
{
	dcoeff = jsarguments[1];
}
if(jsarguments.length>2)
{
	note = jsarguments[2];
}
```

Notez que l'argument **0** est le nom du fichier JavaScript (par exemple, 'globaltask.js'). Par conséquent, de manière réaliste, nous commencerons généralement à examiner les arguments à partie de **1**. Le code ci-dessus vérifie que les arguments existent avant d'essayer d'assigner leurs valeurs à des variables.

L'objet Global

Fermez l'éditeur de l'objet *js* et revenez quelques instants au patch du tutoriel. Dans le coin inférieur gauche, examinez la boîte de *nombre* associée à la boîte de *message* contenant le texte. **Bounce starttime $ 1**. Tapez **2000** dans la boîte de *nombre* et entrez la valeur. Envoyez un rebond en cliquant sur le *button* en haut. Remarquez que le délai entre les rebonds dans tous les objets est plus large qu'avant. Essayez de changer la boîte de *nombre* en une petite valeur. L'intervalle de temps devrait commencer à être plus rapide.

Nous nous attendons à ce que notre boîte de *messages* ait envoyé le message **starttime 2000** à un objet *receive* quelque part dans notre patch Max appelé **bounce**. En fait, il définit la propriété **starttime** d'un objet Global (assigné à répondre au nom **bounce**) sur **2000** dans nos objets *js*. Nous accomplissons ceci en déclarant un objet global dans notre code global:

```
// Global (Max spacename) variables
glob = new Global («bounce»);
glob.starttime = 500;
```

Dans notre code, nous avons créé une variable (**glob**) et l'avons affectée à un nouvel objet Global. L'argument de l'objet global ("**bounce**") est le nom dans l'espace de nom Max qui sera lié à l'objet. Tout message envoyé à **bounce** dans Max tentera de définir les propriétés de l'objet Global en utilisant ce nom. Notez qu'en interne, nous nous référons à l'objet Global par un nom de variable de notre choix (**glob**), et non par le symbole avec lequel Max et notre objet *js* communiquent.

Nous avons ajouté une propriété (**starttime**) à notre objet simplement en l'assignant dans notre bloc global. Désormais, tout message commençant par **starttime** envoyé à **bounce** dans notre patch Max définira cette propriété à ses arguments.

De plus, cet objet est véritablement global, en ce sens que Max peut non seulement le définir depuis l'extérieur d'un objet *js*, mais plusieurs objets *js* partagent l'instance spécifique de cet objet et ses propriétés. Vous pouvez utiliser cette fonctionnalité pour que plusieurs objets *js* partagent des informations, ainsi que pour que Max diffuse des informations vers plusieurs objets *js*.

Résumé

JavaScript vous permet de planifier des événements de manière dynamique à l'aide de l'objet **Task**. Vous créez une tâche et la liez à une fonction qui appelée par la tâche. Vous pouvez activer et annuler la tâche et définir l'intervalle de temps de la tâche et la fréquence à laquelle elle se répète. De plus, en utilisant la propriété **callee** de la fonction appelée par la tâche, vous pouvez définir ces éléments à partir de l'événement programmé lui-même. Toutes les méthodes des objets *js* (qu'elles soient appelées en interne ou par des messages Max) sont exécutées avec une priorité faible dans le planificateur.

JavaScript comporte un certain nombre d'objets de base qui fournissent des fonctionnalités pour les routines de programmation courantes que vous jpouvez trouver nécessaires. L'objet **Math**, par exemple, vous donne accès à une variété de fonctions mathématiques que vous trouverez dans la bibliothèque de mathématiques en C ou dans l'objet Max *expr*.

Les arguments de l'objet *js* sont traités par la propriété **jsarguments** de l'objet. L'objet commence à numéroter ses arguments à **0**, mais le premier argument de *js* est le nom du fichier source qu'il a chargé.

Les objets globaux en JavaScript permettent la communication entre les objets *js* et permettent de définir les propriétés de l'objet directement à partir de l'environnement Max.