Tour d'horizon

1. Usages de Lua

Aucun langage de programmation n'est adapté à tous les développements informatiques. Lua ne fait nullement exception à cette règle. Toutefois, il peut être utilisé dans des situations fort diverses. Ce chapitre cherche à énumérer les avantages et inconvénients de Lua comparés à d'autres langages de programmation courants.

Le slogan de Lua, traduit de l'anglais, était naguère : « Un langage d'extension extensible ». Cela résume bien le positionnement de Lua dans le domaine de la programmation. Si vous y réfléchissez, vous pourrez constater que les avantages et inconvénients de Lua vis-à-vis des autres langages ont presque tous pour origine cette orientation.

Points forts

Lua ne manque pas d'atouts. Certains points forts du langage sont presque uniques et peuvent justifier à eux seuls l'adoption de Lua dans un nouveau développement.

Sveltesse

Une caractéristique très importante de Lua est sa toute petite taille. Si, sur des ordinateurs de bureau actuels, il n'est pas rare de voir des logiciels occuper plusieurs centaines de mégaoctets d'espace disque, les capacités de stockage et de mémoire sont beaucoup plus faibles dans les produits électroniques embarqués. Le compilateur, l'interpréteur et les librairies standards de Lua n'occupent qu'environ 150 kilo-octets une fois compilés. C'est facilement dix fois moins qu'un environnement Perl ou Python réduit à son strict minimum, ou cent fois moins qu'un compilateur C++ moderne en ligne de commande.

Portabilité

Le code source en C est portable. Beaucoup de programmes se targuent d'être portables sous prétexte qu'ils peuvent tourner à la fois sur Windows, MacOS et Linux. Lua va beaucoup plus loin dans sa définition de portabilité, car il peut être compilé sur toute plate-forme possédant un compilateur C respectant la norme ANSI. Le code source respecte même la norme C++98 en plus de C89, ce qui fait qu'il peut être compilé au choix comme un programme C ou C++.

Ainsi, Lua peut être utilisé dans presque tout appareil électronique intégrant un microprocesseur, qu'il soit de type ARM, MIPS, PowerPC ou autre. La taille des mots (32 ou 64

bits), l'ordre des octets dans les mots, le système d'exploitation (voire son absence) sont tout à fait indifférents pour Lua. Bien que normalement le type `double` soit utilisé pour représenter toute valeur numérique, Lua peut être configuré pour utiliser `float` ou `int` comme type numérique, rendant alors possible son utilisation sur un DSP par exemple.

Intégration en C/C++

Lua est écrit sous forme de librairie C et ne contient pas de fonction `main`. Ce qui veut dire que Lua est fondamentalement conçu pour être embarqué à l'intérieur d'une application ou d'un micrologiciel et appelé comme une extension du programme principal. Certes, il existe un programme indépendant nommé `lua` qui permet en apparence d'exécuter des scripts en dehors de toute application. Mais il faut comprendre que cet interpréteur est également une petite application en C (`lua.c`) qui utilise Lua sous forme de librairie.

L'interface de programmation en C est fondamentale pour Lua. Elle ne constitue pas un ajout fait sur le tard comme dans d'autres langages avec leurs différents *bindings*, elle en est son moteur initial. Les librairies standards de Lua s'appuient sur elle, comme d'ailleurs (dans une certaine mesure) la machine virtuelle de l'interpréteur. L'API est donc particulièrement bien pensée et facile à utiliser depuis une application tierce.

Licence libérale

Lua est entièrement gratuit et libre. Il est distribué depuis la version 5.0 sous la licence particulièrement libérale MIT, qui permet d'intégrer ce langage dans tous les types d'applications, qu'elles soient propriétaires, commerciales ou sous licence libre GPL. Il vous est également permis de modifier les sources comme bon vous semble, sans obligation de publier les modifications apportées. L'association *Lua.org* à Rio conserve toutefois la propriété intellectuelle pleine et entière de Lua. Ce point est particulièrement important en milieu industriel où les licences libres sont souvent considérées comme dangereuses pour le logiciel interne : avec Lua, aucun problème de ce type.

Rapidité

Lua est probablement le langage de programmation interprété le plus rapide. Il tourne nettement plus vite que Perl ou Python par exemple. Certes, les langages compilés comme le C ou C++ sont encore nettement plus performants, mais leur usage est différent. Le compilateur de Lua, à savoir la partie du langage chargée de traduire le code source en *bytecode*, est aussi très efficace. Lua n'aura aucun mal à exécuter des scripts composés de plusieurs dizaines de mégaoctets de données.

Mieux encore : il existe depuis quelques années une nouvelle implémentation de Lua nommée *LuaJIT*. Ce logiciel, développé par Mike Pall de façon indépendante de Lua, est certes nettement plus gros et moins portable que l'original. Mais LuaJIT est compatible avec Lua aussi bien sur le plan syntaxique que de l'API, et est incroyablement rapide ! Grâce à son génial compilateur en temps réel et à sa puissante interface FFI, LuaJIT peut parfois battre un code équivalent écrit en C !

Simplicité

La syntaxe de Lua est très simple. Comme vous pourrez vous en rendre compte dans les prochains chapitres, l'essentiel du langage peut être appris en une seule journée. Le nombre d'opérateurs est environ la moitié de celui du C, et il n'y a qu'un minimum de structures grammaticales. Il y a exactement huit types de données que l'on peut affecter à toute variable ou argument de fonction. Parmi ceux-ci, deux types, à savoir *table* et *userdata* sont très versatiles et permettent de construire pratiquement n'importe quelle structure de données. Même les concepts les plus avancés de Lua, comme ceux utilisés pour la programmation modulaire, fonctionnelle ou orientée objet, n'utilisent que des briques élémentaires comme des tables et des fonctions.

Extensibilité

Grâce à son API très bien pensée et aux structures de données versatiles introduites plus haut, Lua est conçu pour être lui-même étendu afin de satisfaire les besoins d'intégration dans une application. L'extension la plus courante est d'exporter vers Lua des fonctions de l'application (en général écrites en C ou C++), empaquetées dans une table. Les fonctions de cette librairie pourront alors être appelées depuis Lua exactement comme une fonction standard.

Il est aussi possible de modifier certains fonctionnements apparemment internes au langage. Par exemple, il est facile de modifier la manière dont Lua charge un module optionnel ou de changer le fonctionnement de la fonction élémentaire `print`. Il est aussi assez simple de créer un environnement d'exécution sécurisé (ou *sandbox*).

Points faibles

La plupart des points faibles de Lua sont le revers de sa portabilité et de sa petite taille. Le langage ne peut rivaliser en fonctionnalités avec des environnements aussi complets que Java par exemple.

Peu de librairies standards

Comme Lua n'utilise strictement que des fonctions normalisées dans ANSI C, la plupart des fonctions des systèmes d'exploitation ne sont pas disponibles. Il n'y a notamment pas de notion de répertoire dans Lua, ni de *socket* ou d'environnement graphique. Ce qui fait qu'il est difficile de réaliser un projet informatique réel en se basant exclusivement sur la distribution officielle. Vous devrez certainement ajouter des modules externes, ou exporter des fonctions de votre application dans Lua.

Librairies tierces éparpillées

Il existe bien de nombreuses librairies et outils pour Lua. La plupart de ces logiciels ont la même licence libérale que Lua lui-même. Mais, contrairement à *CPAN* pour Perl, il n'existe pas de site centralisant et classant tous les modules Lua. Votre meilleur atout si vous avez besoin d'une librairie Lua reste un moteur de recherche.

Heureusement pour vous, toute une partie du livre est consacrée à exposer les librairies tierces que nous avons jugées les plus intéressantes. Nous ferons fréquemment référence à certaines d'entre elles dans des exemples.

Et une certaine avancée a quand même été réalisée ces dernières années. L'outil *LuaRocks* permet de décrire des paquets de distribution Lua sous forme de sources, qui sont automatiquement compilés lors de l'installation, un peu comme avec *CPAN*. Un nombre croissant de modules Lua supportent ce protocole. De plus, des distributions comme LuaDist [http://luadist.org/] et LuaForWindows [http://d-booker.jo.my/luaforwindows] cherchent à pouvoir permettre au programmeur novice d'avoir rapidement un environnement de développement complet.

Il existe aussi un site de type wiki nommé lua-users.org [http://d-booker.jo.my/lua-wiki], qui est ouvert à tout contributeur. Vous y trouverez de nombreux conseils et des extraits de code fort utiles.

Petite communauté

Le nombre de programmeurs utilisant Lua est bien plus faible que celui utilisant Java, C ou C# par exemple. En dehors de l'industrie du jeu vidéo dans laquelle Lua s'est largement imposé, la plupart des informaticiens n'ont jamais entendu parler de ce langage. Cela signifie pour vous qu'il n'existe pas beaucoup de littérature sur Lua, et notamment en français. L'ouvrage que vous avez dans les mains cherche à combler un peu ce manque.

Il est à relever que certains titres de jeux à succès, notamment *World of Warcraft (WoW)*, utilisent Lua comme langage d'extension. Tous les joueurs souhaitant ajouter des niveaux

à leur jeu favori vont donc se frotter à Lua, bien que souvent ils ignorent que le langage existe pour lui-même en dehors du jeu !

Exemples d'utilisation

Vous l'aurez compris, Lua se destine clairement comme un langage embarqué. Cela peut être tout aussi bien dans une application serveur sur un gros ordinateur, dans un jeu vidéo pour téléphone portable comme dans le micrologiciel d'une imprimante. Les prochains chapitres montrent quelques usages divers et variés pour lesquels Lua est particulièrement intéressant, avec chaque fois un petit exemple pour illustrer la situation.

Comme fichier de configuration

Classiquement, sous Unix, la configuration d'un programme se fait sous la forme de fichiers texte qu'il convient de décoder et d'interpréter correctement. À l'origine, sous Windows, les fichiers `.ini` étaient largement utilisés dans ce but. De nos jours, les fichiers au format XML et la base de registre de Windows notamment ont largement supplanté le format texte pur. Néanmoins, utiliser un langage de programmation complet à la place d'un format de données pur comme le XML offre des possibilités de flexibilité largement supérieures. À noter que Lua a été créé, à l'origine, dans ce but : décrire des configurations complexes de façon flexible.

```
-- Configuration d'un client web
server = { host="www.lua.org", port=80, protocol='http' }
basedir = 'c:/MyClientSoftware/'
userdir = basedir.."data/"..os.getenv('USERNAME')
```

Comme base de données

Pour des grandes quantités de données, ou si l'application nécessite une forte sécurité contre la corruption de celles-ci, un moteur de base de données s'impose. Par contre, dans de nombreux autres cas, une base de données pourra être éditée à la main ou générée par un outil externe. Et alors, un format basé sur du texte sera privilégié. Comme dans l'exemple précédent, un fichier en texte pur, séparé par des virgules, ou du XML sont des solutions satisfaisantes. Mais Lua est bien meilleur, grâce à la grande flexibilité de sa syntaxe ainsi qu'à l'extrême rapidité de son compilateur.

```
-- Liste des auteurs de ce livre
authors = {
  { first="Cyril", last="Doillon" },
```

```
    { first="Sylvain", last="Fabre" },
    { first="Philippe", last="Lhoste" },
    { first="Patrick", last="Rapin" },
}
```

Comme langage dédié

Dans certains cas, il est préférable d'utiliser un langage de programmation dédié à une application, au lieu d'un langage généraliste. Imaginons par exemple que l'on souhaite piloter un robot à l'aide d'une ligne de commande. On aimerait alors une syntaxe courte, du genre `marche "1.2m" tourne "90°"`. Eh bien, cet extrait de code est syntaxiquement correct en Lua ! Et même s'il ne l'était pas, il serait beaucoup plus facile de l'interpréter correctement à l'aide d'un langage de script existant plutôt que d'écrire un nouvel interpréteur à partir de zéro.

```
-- Affiche un rectangle et un texte
rectangle { position = { 30, 20 },
  size = { 40, 50 } }
text { text = "Hello World",
  size = { 80, 20 },
  position = { 40, 80} }
```

Comme langage de prototypage

De nombreux programmes utilisent Lua pour accélérer leur développement. Car ainsi, il n'est pas nécessaire de recompiler l'application pour essayer une nouvelle fonctionnalité. Dans les jeux vidéo, la programmation des différents niveaux de jeu se fait généralement avec un langage de script. Les personnes qui imaginent des niveaux supplémentaires ne sont pas nécessairement des programmeurs confirmés. Lua est donc apprécié dans la branche car c'est un langage facile à apprendre, qui de plus est rapide et simple à interfacer dans le jeu.

```
if player:lifeLevel() > 0.3 then
  player:run()
else
  player:walk()
end
```

Comme langage d'extension

Il est naturel d'utiliser Lua pour écrire des greffons à une application. Les applications qui ont opté pour des greffons écrits en C ou C++, même en Java, nécessitent que le

développeur de ceux-ci installe un environnement de compilation *ad hoc*. Pire : pour être vraiment portable, une telle application impose même à l'utilisateur final d'avoir un compilateur pour sa plate-forme pour installer des greffons. Lua permet de s'affranchir d'une telle dépendance : il suffit d'un éditeur de texte ou de copier un fichier dans un certain répertoire pour obtenir une nouvelle fonctionnalité dans l'application.

```
-- Ajoute un menu dans WireShark
register_menu("Lua/Tester serveur Web", function()
    browser_open_url("http://localhost:8080")
  end, MENU_TOOLS_UNSORTED)
```

Comme langage économique

Dans des environnements embarqués restreints en mémoire et en puissance de calcul, dans lesquels on souhaite quand même avoir la flexibilité d'un langage de script complet, Lua est presque un choix obligatoire. Aucun autre interpréteur ne rivalise en taille et en vitesse pour un nombre équivalent de fonctions avec une syntaxe aussi facile à utiliser. C'est pourquoi Lua se retrouve dans des appareils électroniques en tous genres. Il existe aussi des versions minimalistes de Linux qui utilisent des scripts Lua pour remplacer des applications graphiques, normalement écrites en C.

```
-- Convertit en majuscules STDIN vers STDOUT
for L in io.lines() do
  print(L:upper())
end
```

2. Genèse du langage

Contexte

Lua est entièrement développé à l'Université pontificale catholique de Rio de Janeiro (PUC-Rio), dans le groupe Tecgraf traitant de l'informatique graphique. Étonnamment, il se trouve que Lua n'aurait probablement jamais vu le jour sans la politique très protectionniste du Brésil à cette époque. Les compagnies ne pouvaient pas importer de produit informatique, que ce soit logiciel ou matériel, sans apporter de preuves au gouvernement que ce produit ne pouvait pas être fourni par des Brésiliens. Dans ce contexte, Petrobras, la compagnie pétrolière brésilienne, s'est logiquement adressée à l'université pontificale de Rio, renommée, pour leur fournir les logiciels somme toute assez simples dont ils avaient besoin.