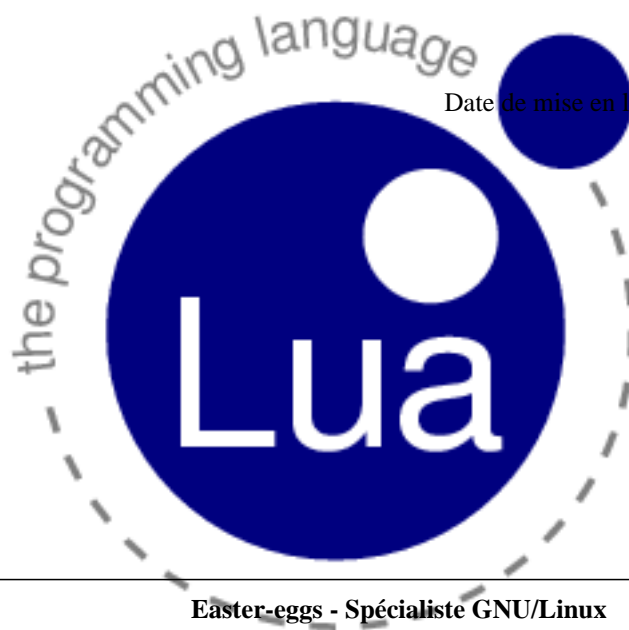


Extrait du Easter-eggs - Spécialiste GNU/Linux

<http://www.easter-eggs.com>

Intégration d'un langage de programmation dans une application avec Lua

- Études de cas - Développement applicatif -



Date de mise en ligne : lundi 19 octobre 2009

Easter-eggs - Spécialiste GNU/Linux

Sommaire

- [Intégration comme système \(...\)](#)
- [Intégration comme API de \(...\)](#)
- [Facilité d'intégration](#)
- [Conclusion](#)

[Lua](#) est un langage de programmation libre, réflexif, impératif et fonctionnel créé en 1993 à l'Université Rio de Janeiro au Brésil.

Lua est écrit en C ANSI, ce qui lui confère une grande portabilité, des systèmes les plus courants (GNU/Linux, Windows, ...) jusqu'au monde de l'embarqué.

Lua est également énormément utilisé dans le monde des jeux video, pour sa portabilité et sa facilité d'intégration avec un programme existant.

Il est utilisé par des projets comme le célèbre jeu World of Warcraft de Blizzard Entertainment ou encore SimCity 4. Il a également été porté sur la console portable Sony PSP et est utilisé pour la programmation de jeux Nintendo DS.

Il est également utilisé par le gestionnaire de fenêtre [awesome](#).

Intégration comme système de configuration

La plupart des logiciels utilisent un système de configuration à base de fichier plat, c'est à dire un système de clé/valeur, éventuellement structuré, pour configurer leurs différents paramètres.

Prenons un exemple, et étudions le cas d'un système de filtrage d'e-mail. Il est très facile de mettre en place une système de clé/valeur structuré pour filtrer un e-mail. En utilisant le format [YAML](#) cela donne quelque chose comme :

```
filtre1:
- Subject: SPAM
target: spam-box/
filtre2:
- To: myaddress@mymail.org
From: paul@mymail.com
target: mails-from-paul/
```

Malheureusement, l'utilisation d'un tel système atteint ses limitations très rapidement.

Imaginons qu'un utilisateur veuille effectuer des actions non listées, comme remplacer le mot « dromadaire » par le mot « fougère ». Cela devient possible si le logiciel fournit une telle directive dans sa configuration, mais le nombre

de directives prévues est forcément limité. On peut remarquer que l'ensemble de ces filtres est très facilement définissable comme un jeu de fonctions du type "fonction(message) return message, maildir".

```
function filtre1(message)
  if message.subject:match("SPAM") then
    return message, "spam-box"
  end
end
```

Puisque le message est également retourné par la fonction de filtrage, il est très simple de le manipuler, et par exemple d'en modifier le contenu, et ce sans rajouter toutes les possibilités et les types de modification au logiciel de filtrage lui-même, ce qui est impossible à obtenir avec YAML.

```
function filtre1(message)
  -- Replace words
  message.content:gsub("dromadaire", "fougère")
  if message.subject:match("SPAM") then
    return message, "spam-box"
  end
end
```

De la même façon, imaginons qu'un utilisateur veuille changer la configuration du logiciel de manière événementielle, c'est-à-dire lorsqu'un événement E se produit. En l'état actuel cela lui est totalement impossible, à moins que le logiciel prenne en charge une directive de configuration pour cet événement.

En reprenant notre exemple ci-dessus, imaginons que l'utilisateur veuille numéroter chaque message, en rajoutant au début du sujet un identifiant numérique.

Le logiciel peut très bien rajouter une directive supplémentaire pour prendre en charge cette fonctionnalité, et modifier le format de son fichier de configuration.

Cependant, en utilisant Lua comme langage de programmation dédié à la configuration, il est très facile pour l'utilisateur de rajouter cette fonctionnalité :

```
message_number = 0

function on_message_delivery(message)
message_number = message_number + 1
end

function filter1(message)
message.subject = message_number .. message.subject
return message
end
```

Le développeur de notre logiciel de filtrage doit uniquement implémenter un système de signaux (ou de « hooks »). Dans l'exemple ci-dessus, la fonction "on_message_delivery" sera automatiquement appelée à chaque remise d'un e-mail dans une boîte mail. En reproduisant ce mécanisme pour tous les événements produits par le logiciel, l'utilisateur sera en mesure de personnaliser son comportement directement depuis le fichier de configuration.

Intégration comme API de programmation

La définition d'une API de programmation en Lua se fait également de façon très simple. Plusieurs systèmes ont déjà utilisé Lua comme langage d'abstraction de plus haut niveau pour développer une application de manière beaucoup plus rapide et concise qu'avec un langage compilé tel que le C.

Le cas typique est celui de programmation d'IHM. Au lieu d'écrire une IHM fixe, il est possible de fournir une API simplifiée de création de widgets et de leur configuration. Cela laisse l'utilisateur libre de définir une IHM et de programmer son comportement de manière dynamique.

Facilité d'intégration

L'intégration de Lua à un outil existant est simple, voire triviale ! La bibliothèque C Lua utilise un système de pile pour interagir avec l'application, rendant son utilisation simple.

```
/** Définition d'une structure de donnée
 * contenant des coordonnées et la taille d'un objet
 */
struct
{
int x, y, width, height;
} geometry;

/** Pousse sur la pile Lua une table contenant des clés x, y, width
 * et height avec les valeurs correspondants.
 * @param L La pile Lua.
 * @param g La geometry de l'objet.
 */
void
lua_pushgeometry(lua_State *L, struct geometry g)
{
/* Creation d'une table vide qui sera sur le haut de la pile */
lua_newtable(L);
/* Ajout de la valeur de 'x' sur le haut de la pile (au dessus de la table */
lua_pushnumber(L, g.x);
/* Attribution de la valeur g.x tout juste poussé sur la table à la clé 'x' dans la table.
 * -2 correspond à l'avant dernier élément (là ou se trouve la table) et "x" au nom de la clé.
 * La valeur attribué est celle du haut de la pile (g.x).
 * lua_setfield retire la valeur g.x de la pile. */
lua_setfield(L, -2, "x");
/* Ajout de la valeur de 'y' */
lua_pushnumber(L, g.y);
/* table[y] = g.y */
lua_setfield(L, -2, "y");
...
}
```

Conclusion

Lua est un langage extrêmement rapide de par son implémentation, et reste très proche du C. Il permet de fournir très facilement une API de configuration et/ou de programmation à l'utilisateur ou au développeur, et permet de prototyper une application aisément.

La taille du code source (17 KSLOC) et du binaire (100 K) lui permet d'être très facilement intégré dans un système embarqué où les contraintes de taille et de temps d'exécution ne permettent pas d'utiliser un langage plus évolué comme [Python](#).