Язык программирования Lua. Учебник для начинающих

Автор	Сообщение
patch_ua	**Заголовок сообщения:** Язык программирования Lua. Учебник для начинающих **Добавлено:** Чт янв 17, 2013 6:40 pm
не в сети **Зарегистрирован:** Чт янв 17, 2013 2:57 pm **Сообщения:** 757	**Введение** Основным мотивом для написания этого текста стало отстутствие на просторах интернета какого-либо внятного учебника по Lua, для чайников и не только, причем на русском языке. Документация на русском есть, а с чего начать - непонятно. Есть хорошая книжка - "Programming in Lua" (причем нужно именно 2-е издание), но ее почему-то даже в электронном виде нигде нет. Мне удалось найти только препринт 1-го издания, и разумеется на английском. После двухгодичной разработки скриптов для хабов на Lua стало ясно, что уж "для чайников" я и сам руководство сделаю. Через некоторое время после осознания этого факта появился черновик, который долгое время оставался совсем убогим и кратким, и вот только сейчас он становится похож на учебник. Отмечу также, что на просторах интернета много различных статей вида "как на Lua сделать что-то эдакое", но полного подробного учебника пока что не наблюдается. Либо встречаются руководства для продвинутых, которые уже что-то знают и хотят "расшИрить и углУбить" (с) свои знания. Итак, приступим-с... Disclaimer: это учебное описание языка Lua, а не полное руководство. Некоторые аспекты излагаются с некоторой вольностью, чтобы не затмевать суть дела. Да простят меня за это великие умы мира сего. ПС Не мое. Взял на big.vip-zone.su Последний раз редактировалось **patch_ua** Чт янв 17, 2013 7:32 pm, всего редактировалось 3 раз(а).
Вернуться к началу	профиль лс
patch_ua	**Заголовок сообщения:** Re: Язык программирования Lua. Учебник для начинающих **Добавлено:** Чт янв 17, 2013 6:41 pm

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Обзор Lua

Lua -- относительно несложный и в то же время мощный язык, часто применяемый как средство расширения функциональности в многих приложениях, играх, и, что нам в данном случае более интересно, в DC-хабах. В частности, он поддерживается в Verlihub (специальным плагином) и в PtokaX. Как и многие другие скриптовые языки, он позволяет выполнять код без предварительной компиляции (т.е. является интерпретируемым). С одной стороны, это часто ускоряет и упрощает разработку (изменил пару строчек, перезапустил, и всё), с другой стороны, некторые разновидности ошибок отлавливаются только в процессе выполнения данного куска кода. Скажем, вы вызываете функцию, которой не существует, или передаёте ей неверное количество параметров: если эта строчка в силу логики работы программы не выполнится, вы про свою ошибку даже не узнаете А вот если выполнится, интерпретатор неприлично выругается. Такого почти не бывает в компилируемых языках, в которых код тщательно проверяется на стадии синтаксической проверки, компиляции или линковки.

В отличие например от Pascal, Lua не является строго типизированным языком. При инициализации переменных им не нужно указывать тип -- он сам разберется, а объявление переменных фактически не требуется, так как объявлением является присвоение значения. Преобразование из одного типа в другой часто происходит неявно, что упрощает читаемость программы. При этом для любой переменной можно в процессе выполнения узнать ее тип с помощьют специального оператора. Более того, тип одной и той же переменной может быть изменен в процессе ее существования при присвоении нового значения (другого типа). Это явление называется динамической типизацией. В Lua (как, например, в javascript), имеется специальное значение nil (аналог null), означающее отсутствие 'содержательных' данных. Как и C/C++, Lua является регистро-чувствительным. Переменная AAA и aaa --- это разные переменные!

Примеры программ в данном тексте и интертрепация

Все приводимые ниже примеры вы можете проверить тут: Lua demo -- вставляете код в форму и жмёте кнопочку Run. Ниже образуется вторая форма с выводом вашей программы. Это возможно благодаря тому, что Lua -- интерпретатор и не требует предварительной компиляции кода. Однако для него есть и компилятор luac, который переваривает программу в некий байт-код, который уже быстрее разбирается интерпретатором, что ускоряет загрузку и выполнение программы. Однако скорость выполнения самого скрипта в компилированном и в обычном виде примерно одинакова, в чем я предлагаю убедиться читателю самостоятельно.

В заголовке слово "интерпретация" не случайно было написано с опечаткой. Дело в том, что формально отделить компилируемые языки от интерпетируемых - занятие соврешенно бесперспективное, и степень "интерпретируемости" - это скорее какая-то вещественная (дробная) характеристика от 0 до 1. Ведь даже программа на ассемблере (или, давайте уж по полной - "машинный" код) тоже в известном смысле интерпретируется процессором! Так что по сути разница лишь в том, кто интерпретирует код -- железяка или программа (скажем виртуальная машина той же Java). С другой стороны, если зашить интерпретатор Lua в

процессор (а почему нет, собственно?), компилятором он от этого не станет

Стандартная библиотека

За образец изложения информации о любом языке программирования я беру книгу Бьерна Страуструпа "The C++ Programming Language", в которой, если я не ошибаюсь, ни разу не упоминается ни одна конкретная операционная система или какая-нибудь библиотека ввода-вывода, которая имеет право иметь различную реализацию (или отсутствовать вовсе!) в различных реализациях языка и в разных ОС. Поэтому я стараюсь минимизировать использование функций, которые могут существенно зависеть от ОС - ведь никто не знает, на каких системах возьмется писать программу кто-нибудь из читателей. Досадное исключение - функция print, которая осуществляет вывод на консоль (а точнее - на стандартный поток вывода stdout), который, к счастью, в том или ином виде присутствует в большинстве ОС.

Следует также отметить и такую особенность Lua. Он часто используется как встроенный язык для написания различных расширений, плагинов и тд. В этих случаях у нас вообще, как правило, нет возможности отлаживать нашу программу с помощью print-ов, так как ее вывод зачастую может подавляться. Например, при отладке скриптов для DC-хабов удобно использовать отладочную печать в виде сообщений пользователям хаба (чат хаба используется как консоль вывода). Но это только один из примеров; если читатель сразу возьмется за применение Lua в качестве embedded-языка, пусть подберет себе способ для вывода отладочных сообщений вместо print.

По тем же соображениям мы вообще не будем рассматривать такие например операции, как ввод данных с клавиатуры, так как их реализация не имеет ничего общего в различных операционных системах. Файловый же ввод-вывод чуть более стандартизован (и самое главное, бывает нужен в приложениях), поэтому его мы всё-таки коснемся в соответствующей главе. То же самое относится к аргументам командной строки программы.

Элементы и особенности языка

Как и в большинстве языков, в Lua есть переменные, операции (арифметические, строковые, и тд), операторы (циклы, ветвление), функции (то есть именованные подпрограммы с параметрами), и библиотека стандартных функций, обеспечивающих взаимодействие программы с ОС и некоторые другие стандартные действия. Оператор цикла for чаще всего используется как конструкция for each и очень применяется для выполнения операций над массивами (таблицами и мета-таблицами), хотя он допускает и классический арифметический вариант с переменной-счётчиком. Циклы while и repeat-until полностью аналогичны большинству языков. Оператор ветвления if допускает секцию elseif, тем самым фактически делает ненужным оператор выбора (case, switch, case of). Как и в C/C++, в Lua нету различий между процедурами и функциями, и те и другие декларируются ключевым словом function, а возвращаемое значение и его тип определяется внутри кода самой функции инструкцией return. И я не сильно ошибусь, если скажу что на этом список ключевых слов языка практически исчерпывается - их

действительно очень немного (разумеется, если не учитывать функции стандартной библиотеки, коих имеется превеликое множество.

В предыдущем абзаце по сути дела не было упомянуто почти ни одного хитроумного слова, не знакомого человеку, который знает уже какой-нибудь язык программирования. За исключением, пожалуй, одного - загадочного слова мета-таблица. Как мы потом узнаем, в них скрывается вся соль Lua. Пока можно сказать, что это прежде всего, обычные таблицы, но кроме хранения данных они позволяют ещё и перегружать стандартные операции для таблиц, как это делается в C++ с помощью конструкции operator(). Если написанное Вам пока не понятно - не обращайте внимания. На некоторое время мы забудем про всю эту жуть и сначала разберем основы языка, которые, во-первых, элементарны, а во-вторых, очень похожи на базовые конструкции в других языках. Я не случайно ссылаюсь частенько на C/C++, так как многие читатели вполне могут его знать, и им будет проще. Что же касается мета-таблиц, то им будет посвящена целая отдельная глава.

Структура кода

В отличие от C++, Pascal и других строгих языков, в Lua исполнение кода программы начинается последовательно, если этот код не является телом функции. Отдельные конструкции языка разделяются пробелом с необязательной точкой с запятой в конце. Я по привычке их ставлю

Пример 0

Код:

```
1 -- это наша первая программа на Lua
2 print("Hello from World!");
3
4 -- а это функция, которая не вызовется
5 function MyFunction()
6 print("Hello from Function!");
7 end
```

При запуске будет выведена только первая строка (Hello from World), так как функцию MyFunction никто не вызовет, и то что написано внутри, не выполнится.

Простейшие примеры

Сейчас мы разберем несколько простеньких программ, на которых я постараюсь доказать вам, что Lua -- это очень просто.

Пример 1: классический Hello world

Код:

```
1 print("Hello World!");
2
```

Я думаю, не надо пояснять, что она выведет;) В Lua не требуются всякого рода "точки входа" программы, как в C или Pascal. Исполнение скриптов начинается с первой строки, за исключением тел функций. Они

выполняются тогда и только тогда, когда к ним происходит обращение. Следующий пример чуть более содержателен:

Пример 2: переменные

Код:

```
1 a = 1;
2 print("Ура, я научился использовать переменные: a = "..a);
```

Что же выведет наша программа в примере 2? Между переменной a и строкой используется операция склейки (по-научному - конкатенации) строк -- две точки подряд. Она выведет текст и значение переменной, то есть число 1. Не правда ли, всё просто?

Тут же следует сказать о том, что в Lua можно не использовать разделитель ; (точку с запятой) для разделения инструкций. Лично мне ее ставить привычнее, к тому же в C++ их всё равно приходится ставить - так зачем переучиваться? Поэтому я буду их использовать в своих примерах. Кроме того, это вносит дополнительную наглядность.

Строковые константы в Lua можно писать как в двойных, так и в одинарных кавычках, как вам удобнее. Если в тексте встречаются кавычки -- можете написать в апострофах, даже не придётся их экранировать символом \, как это приходится делать в C++. Например вот так:

Пример 3: строки

Код:

```
1 strApos = "Строка с 'апострофами' внутри";
2 strQuot = 'Строка с "кавычками" внтури';
```

А как же быть, если нам надо по каким-то причинам написать строку, в которой (о ужас) есть и кавычки, и апострофы? В этом случае поможет только экранирование:

Пример 4: строковый винегрет

Код:

```
1 str = "Строка с \"кавычками\" и 'апострофами' внутри";
```

Если вам нужно осуществить перевод строки, или вставить какой-либо другой служебный символ, пишем так же, как это делается в C/C++.

Пример 5: переносы и табуляция в строках

Код:

```
1 print("А это строка \n с переносами \n строк.\n а вот тут
табуляция \t и ещё \n один перенос.");
```

Будет выведено следующее:

Вывод примера 5

Код:

```
1 А это строка
2  с переносами
3  строк.
4  а вот тут табуляция    и ещё
5  один перенос.
```

На этом мы заканчиваем обзор и переходим к систематическому изложению. Наша программа на ближайшее будущее будет такой: вначале мы обсудим операции и типы данных, потом операторы языка, потом более детально познакомимся с объявлением переменных.

Вернуться к началу профиль лс

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт янв 17, 2013 6:51 pm

Типы данных: Основы

Простые типы

Сейчас мы кратко обсудим основные типы данных в Lua, не углубляясь в детали. Это нам потребуется для того, чтобы можно было сочинять более содержательные примеры. Позже мы вернемся к типам данных и переменным и особенностям их объявления.

Основными (и наиболее часто используемыми) типами данных в Lua являются числа, строки, логические значения, а также массивы и таблицы из этих значений. Явным образом тип данных никак не указывается - интерпретатор определяет его сам, поэтому многие простые конструкции можно писать чисто интуитивно, и это будет правильно.

Типы данных - введение

Код:

```
1 i=1; -- число
2 d=3.1415; -- тоже число
3 sum=i+d; -- сумма двух чисел
4 str="string"; -- строка
```

Больше ничего пока про простые типы я говорить не буду (сознательно) и предлагаю читателю больше пользоваться собственной интуицией при чтении примеров программ. В сложных языках (том же C/C++) это не всегда работает, а вот в Lua - очень даже. Вообще, приятно изучать язык, в котором есть должная степень наглядности кода - он делает именно то, что написано и ничего более, без особых ухищрений со стороны программиста.

Массивы и таблицы

Что такое массив, наверное всем понятно. Это набор данных одного и

того же типа, занумерованный последовательностью целых чисел (индексами) с "шагом" 1. Обычно индексация делается либо от 0 до N-1, где N - количество элементов массива, либо от 1 до N. И тот и другой подход имеет свои плюсы и минусы; в более низкоуровневых языках вроде C/C++ применяется (и оказывается реально удобнее!) индексация с нуля, а вот в Lua "прижилась" индексация массивов с 1. Как мы потом узнаем, это всё не очень принципиально, и при правильном построении программ об этом вообще не приходится думать.

Как объявляются массивы и как с ними работать? Очень просто:

Типы данных - массивы

Код:

```
1 -- способ 1
2 array1={}; -- новый пустой массив
3 array1[1] = 'first element';
4 array1[2] = 'second element';
5
6 -- способ 2
7 array2={[1]='first', [2]='second'};
8
9 print(array1[1], array2[2]);
```

Как видите, всё очень наглядно. Размер массива можно узнать следующим образом:

Код:

```
print(#array1);
```

Более подробно о массивах, их размерах и других тонкостях мы поговорим в отдельной главе, а пока познакомимся с таблицами. Если читатель знаком, например, с C++ и библиотекой STL, то для него таблица "в первом приближении" - это стандартный контейнер std::map. Для "остальных смертных" придётся сказать, что таблица - это ассоциативный массив, представляющий собой множество пар { (key, value) }, для которого определена операция [] (получение элемента value по ключу key). Немного забегая вперед, скажем что массивы в Lua - это на самом деле тоже таблицы с числовыми значениями ключа key от 1 до размера массива.

Вышесказанное показалось вам слишком сложным и непонятным? Ничего страшного, это просто формальное определение интуитивно простых понятий. Сейчас мы рассмотрим пример и всё станет понятно. Например, у нас есть телефонная записная книжка, в которой контактам сопоставлено (для простоты) по одному телефонному номеру. В Lua эту книжку можно было бы задать так:

Типы данных - таблицы

Код:

```
1 Phones = {
```

```
2 ['Вася Пупкин'] = '(495) 504-50-21',
3 ['Изольда Кшыштопопвжецкая'] = '2-12-85-06',
4 ['МЧС'] = '112',
5 }
```

В качестве индексов в таблицах можно использовать не только строки и числа, а вообще всё что угодно, в том числе другие таблицы. Обращаться к элементам можно по этим самым индексам. Само собой, при попытке взять элемент, которого не существует, вам вернется значение nil.

Для таблиц со строковыми ключами, которые записаны латинскими буквами и не содержат в себе пробелов (то есть по сути являются идентификаторами языка), можно использовать упрощенный синтаксис при обращении к элементам таблицы, делая ее похожей на структуру (или класс). А именно:

Таблицы: OOP-style

Код:

```
1 t = {};
2 t.field_x = 10;
3 t.field_y = 20;
4
5 print(t['field_x'], t['field_y']);
```

Здорово, не правда ли? Запись через точку в присвоении выглядит горзадо понятнее, чем в распечатке значений со скобками. Самое главное, у нас появляется возможность двоякой работы с таблицами - как со структурами и как с массивами (ну, разве что с соблюдением более строгих правил для имен полей).

На этом мы временно оставим таблицы в стороне - этого нам будет достаточно для понимания некоторых дальнейших примеров.

Вернуться к началу профиль лс

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт янв 17, 2013 6:56 pm

Операции в Lua

Настало время поговорить о самых простых средствах языка - операциях. Как и во всех остальных языках, в Lua имеется более чем джентльменский набор операций - арифметических, логических и строковых. Начнем разумеется с простого -- с арифметики.

Арифметические операции

Сколько их, арифметических операций? четыре? Неправильно, шесть: сложение, вычитание, умножение, деление, взятие остатка от деления и возведение в степень. Обозначения их стандартны и используются во многих других языках:

Код:

```
+ - * / % ^
```

Приоритеты операций, разумеется, аналогичны обычной арифметике. Но есть и отличия. Например, операция деления целых чисел не является целочисленной (как это происходит в C). Так что выражение

Код:

```
25/2^2
```

будет равно 6.25, а не 6, как в языке C/C++. Lua по сути не делает различий между вещественными и целыми числами. Приятно отметить, что он позволяет работать с достаточно большими целыми числами (в текущих реализациях - до 2^46), не переходя к экспоненциальной форме хранения.

Кроме арифметических операций, имеется ещё некоторое количество функций математической библиотеки, добавляющие нам разного рода округления, стандартные математические функции типа синусов и косинусов, но их мы обсудим позже.

Логические операции

Основные операции "И", "ИЛИ", "НЕ" в Lua записываются как and, or, not и полностью аналогичны другим языкам. Пример:

Пример 6: Железная логика-1

Код:

```
1 if true and false then
2 print("Вот это ерунда!");
3 end
```

Операции сравнения похожи на операции в C/C++. Сравнение здесь пишется двумя знаками равенства ==, а "не равно" -- комбинацией ~= (тильда и знак равенства).

Пример 7: Железная логика-2

Код:

```
1 if 1 ~= 2 then
2 print("Не врет железяка");
3 end
```

Есть ещё один элегантный случай применения булевых операций, кроме обычных логических выражений. Выглядит он так:

Пример 7.5: Использование or

Код:

```
1 number = nil;
2 number = number or 0;
3 print(number);
4 number = 10;
5 number = number or 0;
6 print(number);
```

Первый раз будет напечатано 0, а второй раз -- 10. Таким образом, если один из операндов or равен nil, выражение "x or y" будет равно противоположному операнду. Это удобно для инициализации переменных заведомо корректными значениями (отличными от nil), если заранее значение не известно. Конструкция выглядит гораздо красивее, чем аналогичное сооружение с if.

Строковые операции

Со строками можно делать много всего, но операция для работы с ними всего одна - конкатенация (или, по-кухонному, склейка) двух строк. Надо отметить, что для её обозначения пока что не выработалось общемирового стандарта, поэтому в Lua она обозначается двумя точками (к слову, в PHP для этой цели используется точка, в JavaScript - знак плюс). Пример тривиален, но мы его всё-таки напишем:

Склейка строк

Код:

```
str="Hello".."World";
```

Lua допускает склейку строк с числами - в этом случае последние преобразуются в десятичную запись. Пример:

Пример 8: Склейка строки и числа

Код:

```
1 str="Current Lua version is "..5.14;
```

Но ни в коем случае не пытайтесь склеивать строку с неинициализированной переменной (или, что то же самое, со значением nil):

АнтиПример 9: ошибочная склейка с nil

Код:

```
1 nothing=nil;
2 str="nihil test "..nothing;
```

Если переменная nothing не инициализировалась ранее, программа "вылетит" с ошибкой примерно такого содержания: "attempt to concatenate global 'nothing' (a nil value)". Разумеется, вылетит только во время выполнения, так как заранее никто не знает, будет где-нибудь задано этой переменной значение или нет.

Смешение типов

Любопытный читатель наверняка захочет применить какие-нибудь операции к переменным тех типов, к которым "официально" их применять нельзя. И тут его ждет некоторое разочарование - они будут восприняты языком вполне адекватно и естественно. Например:

Пример 9.5: Смешение типов

Код:

```
1 num1 = '5' + '7'; -- результат - число
2 print(num1); -- 12
3 num2 = '11' * '13' * '7'; -- результат тоже число
4 print(num2); -- 1001
5 str = '1234'..'5678';
6 print(str); -- 12345678
7 str = '1234'..'4444'/2;
8 print(str); -- 12342222
```

В комментариях написаны результаты операций и их типы. Особенно забавна последняя операция, в которой "строковая" константа (по приоритету) делится на число, а потом склеивается с другим числом. Синтаксис это позволяет, но... вообще лучше всегда использовать явное приведение типов, о котором мы расскажем чуть позже.

Операторы языка

Сейчас мы перейдем к обсуждению того, без чего не напишешь ни одной серьезной программы. А именно, об операторах языка для управления логикой работы программы. Как и в любом уважающем себя языке, в Lua есть ветвление (if..then..else) и циклы (for, while, repeat). Их мы сейчас и обсудим.

Отметим, что в Lua нету так называемых "операторных скобок", вроде { ... } в C/C++ и begin .. end в Pascal. Поэтому большинство операторов имеют форму ключевое слово ... end (кстати, забегая вперед, скажем что то же самое относится к объявлению функций). Сейчас мы подробно обсудим каждый из операторов, а начнем с самого часто используемого -- условного оператора if.

Код:

```
if..elseif..elseif..else..end
```

Синтаксис его таков (поскольку в Lua переносы строк не имеют значения, будем для красоты использовать многострочную запись):

Первый, самый простой вариант без else:

if: простой вариант

Код:

```
1 if логическое_выражение then
2 -- ...
3 -- тут идут инструкции, выполняющиеся если выражение истинно
4 -- ...
5 end
```

Вместо многоточий пишите то, что должно быть выполнено, если логическое_выражение может быть (неявно) приведено к логическому true. В Lua значения nil и false воспринимаются как ложь, всё остальное - как истина (даже значение 0). Это немного непривычно для программистов на C и приучает писать сравнения более аккуратно.

Второй вариант, с else:

if: обычный вариант

Код:

```
1 if логическое_выражение then
2 -- тут идут инструкции, выполняющиеся если выражение истинно
3 else
4 -- а тут -- если оно ложно
5 end
```

Если условие верно, выполняется первый блок, если нет -- второй. Альтернативных условных блоков может быть много, в этом случае добавляется elseif:

if: длинный вариант

Код:

```
1 if выражение1 then
2 -- сюда идем, если выражение1 истинно
3 elseif выражение2 then
4 -- сюда идем, если выражение2 истинно
5 elseif выражение3 then
6 -- сюда идем, если выражение3 истинно
7 ...
8 else
9 -- сюда идем, если не выполнилось ни одно из предыдущих условий
10 end
```

TODO: пример на if

Цикл for

Цикл имеет два варианта синтаксиса: арифметический и for-each. Разберем сначала первый, для этого хватит пары примеров:

for: арифметический

Код:

```
1 for i = 1,10,1 do
2 print("Переменная i внутри цикла равна "..i);
3 end
```

Тут переменная пробегает значения от 1 до 10 и программа выводит 10 строчек с её значениями. Третье число означает шаг увеличения переменной (если оно равно 1, его можно не писать вообще). Важная особенность: выражения, определяющие начальное, конечное значение и шаг переменной цикла вычисляются один раз при входе в него. Если потребуется выйти "досрочно" - используйте оператор break, о котором будет рассказано далее.

Арифметическая форма цикла for используется редко ввиду особенностей языка, а точнее ввиду его области наболее частого применения. Гораздо чаще в Lua применяется второй вариант (если он сейчас будет совсем не понятен, ничего страшного - дальше мы ещё будем очень подробно говорить о таблицах, ключах и индексах). Он обычно называется for-each, так как некоторое действие (в данном случае тело цикла) делается для каждого элемента из некоторого множества (в примере - для каждой записи таблицы).

for: модификация for-each

Код:

```
1 -- зададим таблицу из 3 элементов:
2 AdminNicks = {
3 "root",
4 "admin",
5 "master"
6 }
7 -- пробежимся по таблице в цикле:
8 for k,val in pairs(AdminNicks) do
9 print("К таблице в данном случае можно обращаться так: "..val..",
а можно так: "..AdminNicks[k].."."");
10 end
11
```

Программа выведет три строки с упоминанием в них элементов таблицы. Переменная k пробегает индексы таблицы (т.е. массива), а val -- значения элементов по этим индексам. Здесь мы попутно бегло познакомились с очень полезной конструкцией pairs, которая выдаёт ключ (индекс) таблицы и ее элемент. К этому примеру мы ещё вернемся позже.

В качестве более интересного примера для циклов напишем программу, которая выведет все простые числа от 2 до 1000 (делящиеся только на 1 и само себя). Алгоритм самый примитивный, который только может быть (перебор всех делителей).

for: поиск простых чисел

Код:

```
1 function isPrime(x) -- функция проверки простоты
```

```
2 for d = 2, x-1, 1 do
3 if (x % d) == 0 then return false; end
4 end
5 return true
6 end
7
8 for i = 2, 1000, 1 do
9 if isPrime(i) then print(i); end
10 end
11
```

Здесь скобки вокруг x % d стоят исключительно для наглядности. Приоритет операции % (остаток от деления, так же как и в C) выше, чем приоритет сравнения.

Цикл while

Удобство цикла for проявляется тогда, когда мы знаем, сколько раз нам этот цикл крутить. А если нам нужно какое-то действие выполнять, пока верно некоторое условие, удобнее использовать цикл while. Синтаксис его очень простой:

Синтаксис while

Код:

```
1 while логическое_выражение do
2 --операции, выполняемые пока выражение истинно
3 end
```

При входе в цикл значение выражения проверяется на истинность. Если оно ложно, цикл не выполняется ни разу. Если истинно, выполняется первая итерация, и снова проверяется условие. Если оно снова истинно - выполняется вторая итерация, и так далее. Если наше выражение так и останется истинным - цикл будет бесконечным. Наш более содержательный пример - нахождение положительного корня уравнения x^2=2 методом половинного деления на отрезке от 1 до 2. Его значение мы и так знаем: sqrt(2), но мы вычислим его двумя способами и сравним результаты.

Пример использования while

Код:

```
1 function f(x)
2 return x^2 - 2;
3 end
4
5 left=1.0; -- левая граничная точка
6 right=2.0; -- правая граничная точка
7 m = left; -- начальное приближение корня
8
9 while math.abs(f(m)) > 0.000000001 do
10 if f(m) > 0 then
11 right=m;
```

```
12 else
13 left=m;
14 end
15 m = (left+right)/2;
16 end
17 print("Root value: ", m);
18 print("Sqrt(2) : ", math.sqrt(2));
19 print("Difference: ", math.abs(m-math.sqrt(2)));
20
```

Тут, как Вы наверное уже заметили, есть сюрприз в виде неизвестных нам пока функций math.abs и math.sqrt. Знакомые с другими языками разумеется догадались, что это модуль (абсолютное значение) числа и квадратный корень соответственно. Я намеренно включаю в свои примеры некоторые функции из стандартной библиотеки, чтобы читатель постепенно их тоже осваивал по ходу пьесы.

Если выполнить программу, мы увидим, что результаты отличаются только в 11-м знаке. Неплохая точность, не правда ли?

Цикл repeat..until

Цикл repeat..until отличается от цикла с пред-условием (while) тем, что всегда выполняется хотя бы один раз. Синтаксис его аналогичен while с точностью до зеркального отражения:

Синтаксис repeat

Код:

```
1 repeat
2 -- тело цикла
3 until логическое выражение
```

Цикл выполняется до тех пор, пока выражение не станет истинным. Следующий пример печатает первое число, которое больше 2000 и делится на 13:

Пример использования repeat

Код:

```
1 x=2000;
2 repeat
3 x=x+1;
4 until (x % 13) == 0;
5 print(x);
```

Блок do..end

Эта конструкция сама по себе ничего не делает, это просто блок команд, аналогичный {...} в C/C++. Как мы потом узнаем, она влияет на видимость переменных, их глобальность и время жизни. С его помощью удобно

выделять логически разные куски кода, если по каким-то причинам их неудобно выносить в отдельные функции. Переменные, обявленные локальными внутри блока, погибают при выходе из его, поэтому целесообразно применять его например в тех случаях, когда мы имеем дело с "большими" (по объему памяти) переменными, и т.д.

Управляющие конструкции в циклах

Прервать выполнение цикла досрочно можно с помощью оператора break. Его можно использовать только внутри циклов. Синтаксис его использования таков:

Пример использования break

Код:

```
1 i=0;
2 while true do -- организуем бесконечный цикл
3 print("Бесконечный цикл");
4 if i > 10000 then
5 print("Всё, хватит");
6 break;
7 end
8 i=i+1;
9 end
```

Вернуться к началу | профиль | лс

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт янв 17, 2013 7:00 pm

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Переменные: более пристальный взгляд

Как правильно объявлять переменные?

В Lua можно определять переменные практически где угодно. Вернее, поскольку специальное объявление как таковое не требуется, можно вводить переменную именно тогда, когда уже известно ее значение. Это позволяет избежать ошибок с использованием неинициализированных значений. Стиль, навязываемый скажем, языком Pascal, в котором переменные объявляются в специальной секции, совершенно порочен и кроме ошибок ничего не даёт. Кто-то скажет, что так лучше видно, какие в процедуре есть переменные и какие у них типы, однако же никто не знает, используются они вообще в программе или нет, а уж чему равны - тем более не ясно. Поэтому, объявляйте переменную там и только там, где она вам пригодилась.

Глобальные переменные

Код:

```
1 text="some string";
2 more_text="second string"..text;
3 print(more_text);
```

Тут ничего хитрого нет. Объявляется глобальная переменная text, ей задаётся некоторое значение, а потом переменной more_text присваивается склейка строковой константы и значения первой переменной. Вообще, любая переменная в Lua является по умолчанию глобальной. А вот следующий пример уже кому-то может показаться необычным. Во всяком случае, в других языках такие конструкции не работают:

Переменные через запятую

Код:

```
1 var_first, var_second = 1, "text";
2 print(var_first);
3 print(var_second);
```

Здесь будет напечатано сначала 1, а потом text. На первый взгляд не очень понятно, в чем смысл такой конструкции - казалось бы лучше объявить сначала одну переменную, а потом другую. Но этот прием очень часто применяется в более сложных примерах, где используются функции, возвращающие не одно значение, а сразу много. Тогда это становится удобным. А если вам нужно поменять две переменные a и b местами, это можно сделать очень элегантно:

Обмен

Код:

```
a,b=b,a;
```

Не правда ли, красиво? То же самое можно сделать с тремя, четырьмя переменными (например, переставить их по кругу). Разумеется, в таких конструкциях мы должны внимательно следить за порядком переменных, но это уже, как говорится, дело техники. При всём при этом Lua не понимает, например, вот таких вот "извращений в стиле C":

Lua is not a C++

Код:

```
var1=var2=var3=var4=1;
```

Область видимости переменных

Хороший тон программирования предполагает использовать локальные переменные всюду, где это возможно, и и стараться не использовать глобальных переменных вообще. Поэтому теперь мы будем везде стараться объявлять локальные переменные. Делается это с помощью ключевого слова local.

Локальные переменные

Код:

```
1 local var='some string';
```

Как это всё работает? продемонстрируем на примере:

Локальные переменные

Код:

```
1 var="ext variable";
2 if 1 ~= 2 then
3 local var="int variable";
4 end
5 print(var);
```

Как Вы думаете, что напечатает наш пример? Правильно, Вы угадали - он напечатает строку ext variable, потому что переменная внутри оператора if является локальной по отношению к внешней переменной, и значение будет задано у нее, а не переопределено значение внешней переменной. Разумеется, наш пример корректен, поскольку условие верно, и мы попадаем внутрь if-а.

Разумеется, если написать вот так:

Локальные переменные

Код:

```
1 local var="ext variable";
2 if 1 ~= 2 then
3 var="int variable";
4 end
5 print(var);
```

... то переменная у нас тут всего одна и будет изменено именно ее значение, и программа выведет int variable.

В общем случае область видимости переменной - это блок некоторого оператора (if, for, while) блок операторов do..end, тело функции или же вся программа (точнее файл). Обладая этим понятием, можно сформулировать общее правило: среди всех переменных с данным названием, встречающихся в коде, в выражении используется ближайшая в данной области видимости (если расстоянием считать глубину вложенности областей).

Пожалуй, для полноты картины стоит привести ещё один (не очень хороший, но допустимый) пример кода. К сожалению, он потребует объявления функции, но я надеюсь, интуитивно понятен:

Хитроумная видимость и инициализация

Код:

```
1 function create_var()
2 global_var="value";
3 end
4
5 print(global_var);
```

```
6 create_var();
7 print(global_var);
```

Поначалу можно подумать, что будет выведено значение nil два раза, так как "глобальную" переменную global_var никто не инициализирует, а то что делается внутри функции, так и останется в ее пределах. Ан нет: сначала будет напечатано nil, а второй строкой -- слово "value", потому что внутри функции идёт обращение именно к глобальной сущности. Разумеется, стоит нам написать внутри функции заветное слово local в этом присвоении, как ситуация преобразится радикальным образом, и будет выведено два раза слово nil. Вообще, таких неявных конструкций следует избегать, поскольку они часто порождают ошибки.

Осталось заметить, что модификатор local применим так же и к функциям в Lua и влияет на их область видимости (доступности). Это удобно, когда функция заведомо будет использоваться только в рамках данного файла, в котором она описана и не приведет к "засорению" глобального пространства имён.

Типы данных. Динамическая типизация

А какие вообще в Lua бывают типы?

Наверное многие уже поняли, что в Lua имеют право на жизнь как минимум числа (целые и вещественные, между ними нету разницы), строки и таблицы. На самом деле и это тоже не всё. Фокус в том, что функция в Lua - это тоже тип данных, и с функциями можно обращаться почти так же, как с обычными переменными - присваивать одной функции другую функцию, и многое другое. Но про функции у нас ещё будет отдельный длинный разговор, вернемся на землю к "обычным" переменным и типам. Если читатель ещё не забыл введение, там упоминался ещё один специальный тип nil, озанчающий отсутствие какого-либо объекта. Но и на этом список типов не завершается, поскольку в Lua бывают ещё thread-ы (потоки). Но о них мы поговорим позже.

Объявление переменной (точнее, инициализация), как уже говорилось выше, и определяет ее тип в данный момент времени. Почему "в данный момент", а не вообще? Потому что есть динамическая типизация, о которой речь чуть ниже. Вот несколько примеров определений переменных разных типов:

Переменные разных типов

Код:

```
1 local var_int=10;
2 local var_float=3.1415926;
3 local var_str="this is a string";
4 local var_table={};
5 local var_func = function() print("hello from function"); end;
6 local var_nil = nil;
```

Вот по сути и всё, что в Lua допустимо с точки зрения типов. Последняя

запись примера с на самом деле не имеет смысла: любая неинициализированная переменная и так имеет значение . Другое дело что в более сложной программе такое присваивание может вполне себе иметь нетривиальный смысл. Чуть забегая вперед, можно сказать что такая запись ведёт к разрушению данной переменной, если до этого она ссылалась на некоторый объект.

Динамическое определение типа переменной

Теперь мы поясним смысл выражения "в данный момент" из предыдущего параграфа. Сейчас мы повторим наш пример, смело заменив в нем имена переменных на какое-нибудь одно (назовем ее просто var):

Одна переменная разных типов

Код:

```
1 local var=10;
2 print(var, type(var)); -- сейчас это число, будет напечатано '10
number'
3 local var=3.1415;
4 print(var, type(var)); -- сейчас это тоже число, снова будет
напечатано '3.1415 number'
5 local var="text";
6 print(var, type(var)); -- а сейчас это уже строка, будет
напечатано 'text string'
7 local var={};
8 print(var, type(var)); -- мутируем дальше: var стало таблицей,
напечатается 16-ричный адрес и слово 'table'
9 local var = function(a,b) return a+b; end;
10 print(var(1,2), type(var)); -- дальше - больше: теперь var - это
функция, напечатается '3 function'
11 local var = nil;
12 print(type(var)); -- а теперь переменная погибла, напечатается
'nil nil'
```

Проницательный читатель уже давно догадался, что два минуса "--" озанчают строчные комментарии в Lua, а оператор type возвращает тип переменной.

Вернуться к началу профиль лс

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт янв 17, 2013 7:05 pm

Функции

Маленькое лирическое отступление о функциях... Основа читаемости любого кода - это его структурированность. Если написать текст программы "одним куском", его никто не поймет, а его длина в большинстве случаев увеличится во много раз по сравнению с другими реализациями, а скорость работы возрастёт незначительно. Самый простой способ структурировать программу -- разбить её на подпрограммы, снабдив каждую из них именем. Эти подпрограммы в

народе называются процедурами и функциями. Точнее говоря, в разных языках - по-разному, но термин "функция" есть в большинстве языков. Из популярных, пожалуй, только в ассемблере устоялся термин "процедура", поскольку там функции синтаксически не отличаются от процедур. Обычно под "процедурой" подразумевают подпрограмму, которая делает некоторое действие, но не возвращает никаких значений (например, именно так дела обстоят в Pascal/Delphi), а функция - это подпрограмма, имеющая "результат" в виде некоторого объекта, называемого "возвращаемым значением" (return value) функции. А например в языках C и C++ эта грань, напротив, стерта и там все подпрограммы с точки зрения синтаксиса являются функциями. Так... к чему я это всё? А, ну да. Мне просто хотелось сказать что Lua в этом плане похож на C/C++ и там любая подпрограмма - это функция. А теперь обо всём по порядку.

Простые (классические) примеры функций

Синтаксис объявления (описания) функции таков:

Описание функции

Код:

```
1 function function_name(<arg_list>)
2 -- function body
3 end
```

Описание функции начинается с ключевого слова function, далее в скобках перечисляются аргументы функции (только их имена, никаких типов) которые являются локальными переменными внутри функции. Перед ключевым словом function можно, как и в случае переменных, написать слово local, что повлияет на область видимости самой функции. Аргументами функции обычно является последовательность переменных, разделенных запятой. Почему "обычно"? Потому что есть ещё один интересный и полезный случай, который мы разберем чуть ниже.

В отличие, скажем, от C, в Lua нету понятия "протопипа" (заголовка) функции. Она видна везде в своей области видимости, в том числе и выше своего описания.

Выход из функции происходит либо по достижению последней инструкци в функции, либо с помощью оператора return. Если return-а нет, наша функция будет возвращать значение nil (то есть на самом деле ничего возвращать не будет).

Вот пример содержательной функции, которая осуществляет склейку двух строк с проверкой их типов и пустых значений.

Безопасная склейка строк

Код:

```
1 function SafeConcat(a, b)
2 if not a then return b; end
3 if not b then return a; end
4 return a..b;
```

```
5 end
6
7 print(SafeConcat("test1", nil));
8 print(SafeConcat(nil, "test2"));
9 print(SafeConcat("test3.1", "test3.2"));
10 print(SafeConcat(4.1, 4.2));
```

Этой функцией можно склеивать числа, строки, nil-ы -- всё что угодно, ошибки не произойдет. Потом мы узнаем, как такое можно делать более красиво, но это будет потом.

Функции с переменным числом аргументов

Как и в C/C++, в Lua можно задавать функции с переменным числом аргументов. В этом случае они передаются туда как массив с зарезервированным именем arg с индексами от 1 до n, где n=arg.n (буква 'n' -- это тоже на самом деле индекс массива, имеющий строковый тип и значение 'n'. Если это сейчас не очень понятно - ничего страшного, всё встанет на свои места, когда мы разберемся с таблицами, и иожете пока просто это выражение (arg.n) воспринимать как заклинание). Вместо имен параметров функции нужно написать троеточие. Вот пример, который склеивает все свои аргументы (разумеется, кроме nil-ов) в одну строку:

Многоточие

Код:

```
1 function ConcatMultiple(...)
2 local str="";
3 for i=1,arg.n do
4 if arg[i] ~= nil then
5 str=str..arg[i];
6 end
7 end
8 return str;
9 end
10
11 print(ConcatMultiple('a','b',nil,'c','d'));
12
```

Когда пойдет разговор о таблицах, мы ещё вспомним этот пример и реализуем эту же функцию иначе. Но вернемся к нашему многоточию. В нашем примере мы заменили им все аргументы функции. Разумеется, это не всегда удобно, поэтому есть возможность использовать и такие конструкции:

Параметры и многоточие

Код:

```
1 function PrintVarsWithPrefix(prefix, ...)
2 print(prefix..":");
3 for i = 1, arg.n do
4 print(" "..arg[i]);
5 end
```

```
6 end
7
8 local a=1;
9 local b="string";
10 local c=1.5;
11
12 PrintVarsWithPrefix("Variables", a, b, c);
```

Подобные описания функций можно применять, когда некоторые параметры имеют для нас особое значение, а некоторые в известном смысле одинаковые и нам не очень важно, кто из них первый кто последний. Наш последний пример выведет вот что:

Параметры и многоточие

Код:

```
1 Variables:
2  1
3  string
4  1.5
```

Обратите внимание, что нумерация аргументов функции в этом случае остаётся прежней: arg[1] - это первый "динамический" параметр, а не параметр prefix. Поставить многоточие перед именованными аргументами синтаксис Lua, разумеется, запрещает - в этом случае невозможно было бы определить количество динамических и обычных параметров.

Возвращаемые значения функции и оператор return

В отличие от многих других языков, в Lua возвращаемых значений у функции может быть много, и при этом не надо их оформлять в виде какой-нибудь специальной структуры. Просто пишите их через запятую, и в том же порядке присваивайте. Например, напишем функцию, которая возвращает сразу и частное, и остаток от деления двух чисел. Она будет очень простая и будет состоять из единственной интересующей нас строчки с return-ом.

Множественный return

Код:

```
1 function Div(a, b)
2 return math.floor(a/b), a % b;
3 end
4
5 local X=57;
6 local Y=17;
7
8 local quot, res = Div(X, Y);
9 print(quot, res);
10 if quot * Y + res == X then
11 print("All right");
12 end
```

Я думаю, что тут происходит - и так понятно. Для полной ясности остаётся лишь сказать, что math.floor возвращает целую часть числа, округленную вниз.

Разумеется, если нам нужен только один параметр (скажем, частное), можно написать так:

Код:

```
local quot = Div(X,Y);
```

А если нужен только остаток, первый параметр нужно будет тоже куда-то присвоить. Для этого очень часто используют переменную с именем "_":

Код:

```
local _, res = Div(X,Y);
```

Выглядит может быть слегка коряво, но этот приём (и именно такое имя переменной) очень часто применяется в реально существующих проектах на Lua, поэтому изобретать что-то оригинальное в данном случае не хочется. Если Вам нужно добраться до третьего занчения, никто не мешает использовать "мусорную" переменную дважды:

Код:

```
local _,_,var = SomeFunctionReturningThreeParams();
```

Проницательный читатель скорее всего уже догадался, что если функция возвращает меньше параметров, чем мы написали в присвоении, последние (недостающие) будут заполнены значением nil. Никаких ошибок вам интерпретатор при этом не выдаст, а молчаливо сделает то, что Вы ему сказали. В нижеследующем примере сначала напечатается "1 2 3", а потом "5 nil nil":

Множественный return с багом

Код:

```
1 function one(a)
2 return a;
3 end
4
5 local x,y,z=1,2,3;
6 print(x,y,z);
7 x,y,z=one(5);
8 print(x,y,z);
```

Разумеется, количество возвращаемых значений и их типы могут быть непостоянными. Следующий пример может показаться слегка бредовым, но тем не менее любопытным. В нем будет функция, которая возвращает

переданный параметр "как есть", если он строковый, а если передали число - возвращает его же, преобразованное в строковый формат с выравниванием на 10 символов, и второе значение "true" как флаг осуществленной конвертации:

Return разных типов

Код:

```
1 function Convert(var)
2 if type(var) == "string" then
3 return var;
4 end
5 if type(var) == "number" then
6 return string.format("%10d", var), true;
7 end
8 return "Bug in your code!";
9 end
10
11 local r1=Convert(2);
12 print(r1);
13 local r2=Convert(200);
14 print(r2);
15 local r3=Convert(20000);
16 print(r3);
17 local r4=Convert("some string");
18 print(r4);
19
20 print(Convert());
```

Кстати, обратите внимание на последнюю строчку. Там вызывается наша функция Convert без параметров, хотя по идее должна принимать один параметр. Неужели интерпретатор на это ругнется? Конечно же нет! Функция будет вызвана, вот только параметр будет равен nil. В соответствии с тем, что написано в нашей функции, последней строкой будет напечатано "Bug in your code!".

Прочие особенности вызова функций

В Lua допустимы конструкции следующего вида: пусть у вас есть функция, которая принимает (например) два параметра, и вторая функция, которая возвращает два параметра. Можно использовать такую запись:

Передача двух параметров сразу

Код:

```
1 function exchange(a, b)
2 return b, a;
3 end
4
5 function greater(a, b)
6 if math.abs(a) > math.abs(b) then return true; end
7 return false;
8 end
```

```
9
10 local a, b = 1,2;
11 print(greater(exchange(b,a)));
12
```

Программа напечатает false, поскольку наши переменные будут сначала поменяны местами.

Вышеописанный приём можно применять, если "внутренняя" функция (exchange в нашем примере) стоит на последнем месте в списке параметров. Если это не так - из нее будет взят только один параметр. В следующем примере одна из строчек программы выполнится правильно, а вторая сломается на арифметической операции:

Передача многих параметров сразу - особенности

Код:

```
1 local a1,a2,a3,a4=1,2,3,4;
2
3 function sum4(a,b,c,d)
4 print(a+b+c+d);
5 end
6
7 function nothing(x,y,z)
8 return x,y,z;
9 end
10
11 sum4(a1, nothing(a2,a3,a4)); -- печатает 10
12 sum4(nothing(a1,a2,a3), a4); -- выдаёт ошибку
```

Передача параметров по ссылке и по значению

Обычные типы (строки, числа, логические переменные) в Lua передаются в функции по значению (то есть как обычно). Параметр такого типа является локальной переменной внутри функции, его можно менять и при этом значение той переменной, которую мы передавали в функцию, не изменится. Следующий пример это подтверждает:

Передача параметров по значению

Код:

```
1 function TryToChangeThem(int, float, bool, str)
2 int = -int;
3 float = float/2;
4 bool = not bool;
5 str = str..str;
6 end
7
8 local i=1;
9 local f=5;
10 local b = false;
11 local s = "test";
```

```
12 TryToChangeThem(i, f, b, s);
13 print(i, f, b, s);
```

Здесь будут напечатаны именно те значения, которые были изначально присвоены этим переменным. Но есть и другие типы, для которых всё вышесказанное неверно. Среди тех, про которые мы уже что-то знаем, - это таблицы и функции. Есть ещё два типа, про которые пока мне говорить не хочется, чтобы не пугать читателя - с ними мы познакомимся позже.

Что же касается таблиц, то приведем простой пример, хотя что такое таблица, я ещё детально не объяснял (рекомендуется вспомнить вот этот пример. По-кухонному в данном примере можно считать, что таблица - это просто некоторая структура данных. В нашем случае это будет пара чисел (скажем, точка на плоскости), а функция по сути описывает некое геометрическое преобразование этой плоскости.

Передача параметров по ссылке

Код:

```
1 function Transform(point)
2 point.x = point.x / 2;
3 point.y = point.y * 2;
4 end
5
6 p = {
7 x = 5,
8 y = 6
9 };
10
11 print(p.x, p.y);
12 Transform(p);
13 print(p.x, p.y);
```

Всё дело в том, что когда мы передаём в функцию таблицу, на самом деле туда передаётся только ее адрес, никакого копирования данных не происходит. То же самое, кстати, происходит при операции присвоения на таблицах.

Все эти тонкости надо понимать, чтобы не попортить свои же собственные данные (скажем, "скопировать" одну таблицу в другую и потом начать ее менять, думая что копия-то сохранилась). В других языках выбор типа передачи параметров лежит на пользователе (т.е. программисте), а тут - неявно в зависимости от типа, и за этим вообще говоря надо следить. С той же осторожностью следует обращаться с функциями.

Функция как обычный тип данных

Как уже было сказано выше, функция в Lua -- это такой же тип данных, как число или строка. Поэтому допустимы такие конструкции:

Переменная-функция

Код:

```
1 local sum = function(x,y) return x+y; end
2 print(sum(1,2));
```

Здесь пока ничего необчного нет, кроме того что ключевое слово function и имя самой функции (sum) поменялись местами. Теперь никто не мешает "скопировать" эту функцию в другую переменную (то есть на самом деле просто дать ей альтернативное имя, так как никакого физического копирования кода не происходит), например так:

Код:

```
local sum2 = sum;
```

После этого можно использовать функцию sum2 наравне с sum - она будет делать в точности то же самое.

А вот теперь мы напишем пример посложнее. Он будет осуществлять обычную сортировку массива, который состоит либо из строк, либо из чисел. Причем правило сортировки такое: элемент a1 считается меньше элемента a2, если:

a1 - число, а a2 -- строка;
a1 и a2 числа, и a1 меньше a2;
a1 и a2 строки, и длина a1 меньше длины a2;
Вот такая вот изощренная сортировка. Можно было бы написать одну функцию сравнения элементов, рассмотрев в ней все три случая. Но мы поступим иначе - мы напишем три разных функции сравнения для каждого из случаев, и в зависимости от типов будем выбирать одну из трех.

Мегасортировка

Код:

```
1 local a={"lua",100,1,"scripting",2,45,23,"is very
fun!",52,"super",56,"a",4,6,70,"power!",34,"has"};
2
3 function less_num(p,q)
4 if a[p] < a[q] then return true; end -- number is always less
than string
5 return false;
6 end
7
8 function less_str(p,q)
9 if string.len(a[p]) < string.len(a[q]) then return true; end
10 return false;
11 end
12
13 function less_mixed(p,q)
14 if type(a[p]) == type(a[q]) then error("shit happens"); end
15 if type(a[p]) == "number" then return true; end -- number is
always less than string
16 return false;
```

```
17 end
18
19 function getSortFunction(p, q)
20 if type(a[p]) == "number" and type(a[q]) == "number" then
21 return less_num;
22 elseif type(a[p]) == "string" and type(a[q]) == "string" then
23 return less_str;
24 end
25 return less_mixed;
26 end
27
28 function less(p,q) -- общая функция сравнения двух элементов
(оператор "меньше")
29 local sort_function = getSortFunction(p, q); -- получаем функцию
для сравнения
30 if sort_function(p, q) then return true; end
31 return false;
32 end
33
34 function getMin(s)
35 local l=#a;
36 local m = s;
37 for j=s+1,l do
38 if less(j,m) then
39 m=j;
40 end
41 end
42 return m;
43 end
44
45 function megaSort()
46 local l = #a;
47 for i=1,l do
48 local m=getMin(i);
49 if m ~= i then
50 a[m],a[i] = a[i],a[m];
51 end
52 end
53 end
54
55 printArray = function()
56 local s="";
57 for i=1,#a do s=s.." "..a[i]; end
58 print(s);
59 end
60
61 printArray();
62 megaSort();
63 printArray();
64
```

Когда мы дойдем до изучения таблиц, мы узнаем, что всё то же самое можно было бы сделать гораздо короче и красивее. Мне просто хотелось привести пример относительно большой содержательной программы. На

этом мы завершаем знакомство с функциями и переходим к самому интересному - таблицам.

Вернуться к началу профиль лс

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт янв 17, 2013 7:14 pm

Таблицы и мета-таблицы

Начнем с маленького в меру лирического отступления о том, откуда вообще берется такая суета вокруг таблиц и почему они важны. Любой язык программирования (точнее, сама программа) в известном смысле призван моделировать те процессы, которые происходят вокруг нас, систематизировать и перерабатывать какие-то данные, которые у нас имеются, и т.д. Так вот когда данных становится много и они однотипны, образуются последовательности, таблицы, массивы, матрицы, кубы и так далее. Причем очень часто наши данные не имеют каких-либо ярко выраженных "номеров", которые можно было бы использовать в качестве индексов массива. Или же, номера имеются, но совсем не по порядку или же среди них встречаются вещественные (т.е. не целые) значения. Обычного массива в этом случае не организуешь - он будет либо слишком разреженным, либо неудобным для использования. Каков же выход из этой ситуации? Одно из решений, которое резко упрощает нашу модель - это ассоциативный массив, в котором ключом (индексом) может являться любая сущность, для которой определена операция сравнения (а для хорошей быстрой реализации - ещё и оператор "А меньше Б"). В ассоциативных массивах (то есть std::map-ах в терминологии C++) мы получаем возможность нумеровать элементы любым классом, для которого определены вышеупомянутые операции. Почти то же самое мы имеем и в Lua. Наиболее часто приходится индексировать таблицы числами и строками (а если немного слукавить, то по сути других базовых типов и не бывает в природе).

Отметим также и такой немаловажный момент: эффективная реализация таблиц (ассоциативных массивов) позволяет удобно работать с базами данных, поскольку там структура хранения данных именно такова - у большинства таблиц базы есть ключ (или ключи, что не так важно), которому соответствует ровно одна строка (row) таблицы. Чтобы написать на C++ программу, в которой можно было бы легко и удобно оперировать с записями в базе, нам пришлось бы как минимум все таблицы нашей базы реализовать в виде классов. А в Lua этого делать не надо - всё уже сделано до нас! Просто берем и пользуемся. Кстати, библиотека для Lua, которая реализует работу с БД, тоже существует (LuaSQL), и она в разы проще, чем аналогичный API для C++.

Таблицы

Если читатель ещё не забыл параграф о таблицах в разделе "Типы данных", то он наверняка помнит, что новая пустая таблица конструируется с помощью фигурных скобок:

Код:

```
tab={};
```

Отметим, что в данном случае в памяти создаётся новый объект и переменной tab присваивается указатель на него. Если мы захотим присвоить одну таблицу другой подобным присвоением - в этом случае мы получим не копию, а два указателя на одно и то же место.

Если же мы знаем, чем заполнять таблицу, можно написать и так:

Код:

```
tab={"text", 1, 2, "other text", 3};
```

В этом случае элементы будут автоматически пронумерованы числами, начиная с 1. Давайте в этом наглядно убедимся с помощью простой программы:

Автонумерация

Код:

```
1 tab={"text", 1, 2, "other text", 3};
2 for i,v in pairs(tab) do
3 print(i, v);
4 end
```

Выполните её, и убедитесь, что порядок элементов сохранился, а индексы будут от 1 до 5. Фактически, тут мы получили обычный массив. Для полной ясности надо разобрать подобнее оператор for и конструкцию pairs, чему посвящен следующий параграф, а пока что посмотрим, какие ещё есть способы инициализации таблиц.

Особенности национальной индексации

Индексы можно указывать явно, в данном примере результат будет в точности такой же как и в предыдущей программе:

Явная нумерация

Код:

```
1 tab={
2 [1]="text",
3 [2]=1,
4 [3]=2,
5 [4]="other text",
6 [5]=3
7 }
8
```

А вот пример с индексами разных типов:

Смешанная индексация

Код:

```
1 tab={
2 [1]="text",
3 [2]=1,
4 ["key"]="value"
5 };
6 for i, v in pairs(tab) do
7 print("Key is: ", i, "Value is: ", v, type(i));
8 end
```

Однако лучше не злоупотреблять смешанной индексацией без необходимости, поскольку это может привести к ошибкам. Засада заключается в том, что Lua различает строковые и числовые индексы, и поэтому t[1] не равно t["1"]. Вот пример:

Засада с типами

Код:

```
1 tab_auto={"value"};
2 print(tab_auto[1], tab_auto["1"]);
3
4 tab_man={[1]="value"};
5 print(tab_man[1], tab_man["1"]);
6
7 tab_str={["1"]="value"};
8 print(tab_str[1], tab_str["1"]);
```

Программа печатает вот что:

Код:

```
value   nil
value   nil
nil   value
```

Таким образом, правило простое: какой индекс мы выбрали - по такому и следует обращаться, в случае индексов таблиц автоматической конвертации типов нет.

Тут стоит сказать ещё про одну засаду. Следующий пример относится скорее к категории антипримеров, то есть "как не надо делать". А именно, не стоит использовать одновременно явную и неявную индексацию массивов.

Ошибка инициализации таблицы

Код:

```
1 tab={
2 [1]="number one...",
3 [2]="number two...",
4 "number three...",
```

```
5 "...etc"
6 }
7 for i, v in pairs(tab) do
8 print("Key: ", i, "Val: ", v, "Type: ", type(i));
9 end
```

В этой программе первые два элемента просто погибнут, так как они будут замещены автоматически пронумерованными 3-м и 4-м элементом. В итоге таблица будет из 2 строк, а не из четырех.

Таблицы как структуры

Есть ещё один замечательный способ задания индексов таблиц, который наверняка понравится любителям ООП. Можно использовать такую запись:

Структуры

Код:

```
1 complex = {
2 re=1,
3 im=2
4 };
5 print(complex.re, complex.im);
6 print(complex["re"], complex["im"]);
7
```

В двух строках будет выведено одно и то же, так как запись table.key и table["key"] означает одно и то же. Согласитесь, что первая конструкция намного симпатичнее второй.

for, pairs и ipairs

Для перечисления элементов таблиц нам уже не годится арифметический цикл for, так как вместо номеров (индексов) могут быть и строки, и таблицы, и числа, и функции. Для таких случаев есть оператор in, который умеет перечислять все элементы массива или таблицы в некотором порядке. А в каком именно порядке - это может зависеть от реализации нашего интерпретатора, стандарта на этот счёт не существует. Можно только лишь гарантировать, что он переберёт все элементы. Множество элементов, по которому мы будем бегать, можно получить с помощью оператора pairs. Он будет выдавать нам по очереди все пары [индекс, значение] нашей таблицы (а точнее говоря - [ключ, значение], так как в случае строковых ключей таблицы слово "индекс" не очень уместно).

Распечатка всех значений таблицы

Код:

```
1 tab={"it", "is", "a", "Table"};
2 for i, v in pairs(tab) do
3 print("Key is: ", i, "Value is: ", v);
4 end
```

Распечатка всех значений таблицы (вывод)

Код:

```
1 Key is: 1Value is: it
2 Key is: 2Value is: is
3 Key is: 3Value is: a
4 Key is: 4Value is: Table
```

Я думаю, теперь читателю стали полностью понятны примеры, которые приводились выше в предыдущих параграфах.

Если мы захотим перебрать только числовые индексы таблиц, нам поможет конструкция ipairs. Работает она совершенно аналогично:

Распечатка значений таблицы с числовыми индексами

Код:

```
1 tab={
2 [1]="text",
3 ["key"]="value",
4 [2]="other text",
5 [3]=3
6 }
7 print("Number Pairs");
8 for i, v in ipairs(tab) do
9 print("Key is: ", i, "Value is ", v);
10 end
11 print("All Pairs");
12 for i, v in pairs(tab) do
13 print("Key is: ", i, "Value is ", v);
14 end
```

Размер таблицы, количество элементов, обход всех элементов

Вообще говоря, понятие "размера" в Lua применимо только к массивам, то есть к таблицам с числовыми индексами. Размером массива tab называется индекс i максимального элемента, такого что tab[i] ~= nil, а tab[i+1] = nil (и при этом все элементы с меньшими номерами также не нулевые). Одним словом, размер - это длина непрерывной последовательности непустых индексов от начала масива. Размер можно получить с помощью операции "#":

Получение размера таблицы

Код:

```
1 tab={"array", "length", "sample"};
2 local size=#tab;
3 print(size);
```

Что будет напечатано? Конечно же, число 3. А вот пример похитрее:

Получение размера таблицы с дыркой

Код:

```
1 tab={[1]="array", [2]="length", [3]="sample", [5]="hole between
indexes"};
2 print(#tab);
```

В соответствии с определением длины, тут тоже будет напечатано число 3.

Для произвольной таблицы длина вообще не определена. Узнать количество элементов в ней можно, лишь обойдя ее полностью по уже изученной схеме циклом for.

Стандартная библиотека table

Для удобной работы с массивами в Lua есть стандартная библиотека table. Там есть весьма часто используемые функции, например добавление и удаление элементов, сортировка и прочее. Следующий пример считывает построчно файл программы (то есть самой себя) и складывает в массив, а потом печатает ее:

Нечестный способ распечатать собственный текст

Код:

```
1 local selftext={};
2 for line in io.lines(arg[0]) do
3 table.insert(selftext, line);
4 end
5 for _,line in pairs(selftext) do
6 print(line);
7 end
```

Функция table.insert имеет два обязательных параметра (таблицу и новый элемент), и необязательный параметр position, указывающий куда этот элемент вставлять. Если он не указан, добавление происходит в конец таблицы. Вот та же самая программа, печатающая себя вверх ногами:

Вверх дном

Код:

```
1 local selftext={};
2 for line in io.lines(arg[0]) do
3 table.insert(selftext, 1, line);
4 end
5 for _,line in pairs(selftext) do
6 print(line);
7 end
```

Помните, в самом начале у нас была программа, печатающая простые числа? Вот ее более быстрая реализация с помощью таблиц:

Ускоренные простые числа

Код:

```
1 local primes={2,3};
2 local n=10000;
3
4 function isPrime(x)
5 for _,p in pairs(primes) do
6 if x%p == 0 then return false; end
7 if p > x/2 then break; end
8 end
9 return true;
10 end
11
12 for i = 5,n,2 do
13 if isPrime(i) then table.insert(primes,i); end
14 end
15
16 for i,v in pairs(primes) do
17 print(v);
18 end
19
```

То, что оно работает быстрее, предлагается проверить читателю. А ещё ему предлагается написать аналогичный код, скажем, на C++ и посмотреть, на сколько ему удастся обогнать Lua. Таблицы реализованы в Lua очень эффективно, и проигрывают C-шным реализациям не так уж сильно (иногда "всего" в 2-3 раза, что для интерпретируемого языка, согласитесь, очень нехило). У других скриптовых языков, например у Python в среднем этот коэффициент существенно выше, хотя его спасает очень богатая стандартная библиотека, которая само собой реализована на C и потому "кишки" работают быстро.

Ещё одна операция, которая часто используется в прикладных программах, - это сортировка. Lua умеет сортировать таблицы, если ему объяснить, как сравнивать два элемента. Объяснение делается с помощью функции сравнения, которую надо передать стандартной библиотеке. Если ее не указать, он будет пытаться сравнивать элементы стандартным оператором <, и в простых случаях оно даже сработает. Вот примеры:

Сортировка массива

Код:

```
1 local array_int={3,45,6,234,653,2,51,43,32};
2 table.sort(array_int);
3 print(table.concat(array_int, ', '));
4 local array_string={"lua", "is", "a", "great", "language"};
5 table.sort(array_string);
6 print(table.concat(array_string, ', '));
```

Вернуться к началу профиль лс

talkincat

не в сети

junior developer

Зарегистрирован: Пн май 20, 2013 4:36 pm
Сообщения: 80

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Пн июн 03, 2013 9:13 pm

Собственно, вот второе издание, на английском конечно
Programming in Lua, Second Edition

Год: 2006
Автор: Roberto Ierusalimschy
Жанр: Учебное пособие
Издательство: Lua.org
ISBN: 85-903798-2-5
Язык: Английский
Формат: PDF
Качество: Изначально компьютерное (eBook)
Количество страниц: 329

http://rutracker.org/forum/viewtopic.php?t=3682281

Вернуться к началу профиль лс

Indomito

не в сети

junior developer

Зарегистрирован: Чт май 29, 2014 5:11 am
Сообщения: 6
Откуда: Москва

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт май 29, 2014 5:42 am

patch_ua

> **Цитата:**
>
> Disclaimer: это учебное описание языка Lua, а не полное руководство. Некоторые аспекты излагаются с некоторой вольностью, чтобы не затмевать суть дела. Да простят меня за это великие умы мира сего.
>
> ПС Не мое. Взял на big.vip-zone.su

а можно файл то?
Я войти/скачать не могу - нужен Лог+Пасс

_____
"На каждое действие есть равная ему противодействующая критика."
Постулат Харриссона

Вернуться к началу профиль лс

patch_ua

не в сети

Зарегистрирован: Чт янв 17, 2013 2:57 pm
Сообщения: 757

Заголовок сообщения: Re: Язык программирования Lua. Учебник для начинающих
Добавлено: Чт май 29, 2014 7:06 pm

> **Indomito писал(а):**
>
> patch_ua
>
> > **Цитата:**
> >
> > Disclaimer: это учебное описание языка Lua, а не полное руководство. Некоторые аспекты излагаются с некоторой вольностью, чтобы не затмевать суть дела. Да простят меня за это великие умы мира сего.

> ПС Не мое. Взял на big.vip-zone.su
>
> а можно файл то?
> Я войти/скачать не могу - нужен Лог+Пасс

уже нету файла(

Вернуться к началу профиль лс

Показать сообщения за: Все сообщения Поле сортировки Время размещения по возрастанию Перейти

новая тема ответить **Страница 1 из 2** [Сообщений: 16] **На страницу 1, 2 След.**

Список форумов » **FAQ, инструкции, книги по LUA и QLUA** Часовой пояс: UTC + 2 часа

Кто сейчас на конференции

Зарегистрированные пользователи: **Bing [Bot]**, **Google Adsense [Bot]**

Вы **не можете** начинать темы
Вы **не можете** отвечать на сообщения
Вы **не можете** редактировать свои сообщения
Вы **не можете** удалять свои сообщения
Вы **не можете** добавлять вложения

Найти: Перейти Перейти: FAQ, инструкции, книги по LUA и QLUA Перейти