Utilizando o Lua Xml Parser para leitura simples de arquivos xml em aplicações NCL/Lua

by Johnny Moreira Gomes

Utilizarei este espaço para expor uma maneira simples de se acessar o conteúdo de arquivos xml através do uso da biblioteca Lua Xml Parser, que pode ser obtida através do link http://lua-users.org/wiki/LuaXml . Para versões do Lua a partir de 5.0 é preciso utilizar a versão modificada da biblioteca, encontrada juntamente com a nossa aplicação de exemplo, no arquivo "LuaXML.rar". Grande parte dos conhecimentos abordados aqui, bem como a versão adaptada da biblioteca, foram extraídos da aplicação "Leitor de Rss para Tv Digital" do portal de Manoel Campos (http://ncluarss.manoelcampos.com).

Disponibilizo aqui uma aplicação Ncl/Lua simples e rica em comentários explicativos que extrai dados obtidos do arquivo resposta.xml. O link para a aplicação é http://www.ufjf.br/lapic/files/2010/04/TutorialXmlLua.zip.

A biblioteca nos oferece vários métodos para se criar estruturas que representem o xml em questão. O método que utilizaremos e que cobre a maior parte das necessidades é o "SimpleTreeHandler".

O código abaixo demonstra como utilizar esse método para ler o arquivo xml e criar a table lua correspondente.

```
dofile("LuaXML/xml.lua")
dofile("LuaXML/handler.lua")

---Instancia o objeto que e responsavel por
---armazenar o XML em forma de uma table lua
local tableXml = simpleTreeHandler()

---Nome do arquivo XML a ser lido pelo nosso aplicativo
local FILE_NAME = "resposta.xml"

--Abre o arquivo XML
local f, e = io.open(FILE_NAME, "r")
if f then
      --Passa o conteudo do arquivo para uma string
      local xmltext = f:read("*a")

      --Instancia o objeto que faz a conversao de XML para uma table lua
      local xmlparser = xmlParser(tableXml)
      xmlparser:parse(xmltext)

    --Agora a variavel tableXml e um table lua com o nosso xml embutido
      ...
end
```

Agora que possuímos uma table lua contendo nosso xml estruturado, precisamos entender como se dá essa representação. Sabemos que uma table lua nada mais é do que uma estrutura de dados

dinâmica que armazena vários tipos de dados, incluindo vetores. Esses dados podem ou não se apresentarem de forma hierárquica sob os mais diversos tipos de indexação. Para mostrar como está esquematizada a hierarquia do nosso tableXml, consideremos o segmento de xml abaixo:

```
<people>
    <person id=”1”>
        <name>João Jamil</name>
        <age>22</age>
    </person>
</people>
```

Em nosso tableXml, o conteúdo da tag “name” mostrada acima é acessado da seguinte forma:

tableXml.root.people.person.name
–Retorna “João Jamil”
--Note que root representa a raiz do arquivo xml

Com isso, percebemos que nosso table lua possui campos agrupados “um dentro do outro”, cada um representando um nó ou uma tag de nosso xml. Os campos só apresentarão o valor de texto caso o nó correspondente não possua nós filhos.

Considere agora o segmento de xml:

```
<people>
    <person id=”1”>
        <name>João Jamil</name>
        <age>22</age>
    </person>

    <person id=”2”>
        <name>Maria Milecem</name>
        <age>25</age>
    </person>
</people>
```

Quando considerarmos o campo tableXml.root.people.person não poderemos acessar o valor “João Jamil” como fizemos anteriormente. Isso acontece porque há dois nós filhos de “people” que atendem pelo nome de “person”. Neste caso, tableXml.root.people.person é um vetor contendo os dois nós chamados “person”, indexados na ordem em que aparecem no arquivo xml, começando por 1. Logo:

tableXml.root.people.person[1].name
–Retorna “João Jamil”

tableXml.root.people.person[2].name
–Retorna “Maria Milecem”

Pode acontecer de não sabermos quantos filhos de “people” são chamados “person”, ou seja, podemos não saber quantas pessoas estão registradas no nosso arquivo xml. Para isso, devemos verificar se existe alguma pessoa. Caso positivo, devemos verificar se há mais de uma pessoa, pois nesse caso deveremos tratar o nosso campo como um vetor.

```
If (tableXml.root.people.person~=nil) then –Testa de há pelo menos uma pessoa
    if (#tableXml.root.people.person==0) then
    –Se o tamanho do suposto vetor for zero, indica que o campo não é um vetor
    --Podemos afirmar isso porque aqui já sabemos que há pelo menos uma pessoa
        print(tableXml.root.people.person.name) –Nome da única pessoa do arquivo

    else
    --É um vetor com mais de uma pessoa
        local i
        while i <= #tableXml.root.people.person do
            print(tableXml.root.people.person[i].name)
            i = i + 1
        end
    end
end
```

Até agora vimos como acessar valores internos a tags. Veremos agora como acessar um atributo de um nó. Considere novamente o fragmento:

```
<people>
    <person id="1">
        <name>João Jamil</name>
        <age>22</age>
    </person>
</people>
```

Para acessarmos o id de João Jamil, façamos:

tableXml.root.people.person._attr.id

Se houvesse mais de uma pessoa:

tableXml.root.people.person[i]._attr.id

O campo _attr está presente em qualquer nó de nosso xml e nos transfere ao escopo dos atributos referentes a esse nó. Para acessa-lo basta escrever …**._attr.nome_do_atributo**.

Terminamos por aqui este nosso breve e simples tutorial. Acredito que este método atenderá a maior parte das necessidades em leitura de xml sob aplicações Ncl/Lua. Um bom proveito!

Johnny Moreira Gomes

Aluno do curso de Ciências Exatas da Universidade Federal de Juiz de Fora (UFJF) e Bolsista do Grupo de Tv Digital do Laboratório de Aplicações e Inovação em Computação (LApIC) da UFJF