Az XML és a Perl (4. rész)

Az XML-értelmezés DOMináns módszere

Az XML-értelmezés tekintetében a legnépszerűbb módszerek közé tartozik a W3C által megalkotott Document Object Model (DOM). A DOM-értelmező az XML dokumentumot egy objektummodellé fordítja le – ezzel teszi lehetővé a program számára a véletlen elérést és módosítást a dokumentumban. Az objektummodell azt jelenti, hogy az értelmező egy objektumot hoz létre, az úgynevezett dokumentumobjektumot, amely a dokumentumot jelképezi. Mint minden objektumnak, ennek is vannak tulajdonságai, ezek a dokumentum alkotórészeit jelképezik; illetve elemfüggvényei, amelyekkel az utóbbiakat lehet elérni és módosítani. Minden alkotórész-tulajdonság önmagában is egy objektum, és további objektumok lehetnek a tulajdonságai között. Az alkotórész-objektumok felépítése megfelel a dokumentum felépítésének.
A DOM eredetileg egy API-ként fogant meg a HTML-be ágyazott parancsállományok dokumentumeléréshez és -módosításhoz való támogatására, azzal a céllal, hogy a webböngészőben „dinamikus tartalom” jelenhessen meg. Mint az később kiderült, a DOM kiváló módszere az XML-dokumentumok modellezésének. Ez vezetett a szabványos DOM-felület megszületéséhez, amit a W3C ajánlás formájában fogalmazott meg. Majdnem az összes XML-értelmezőnek van DOM-felülete, mi több, a Microsoft XML-feldolgozó eszközei kizárólag DOM-felületet kínálnak. A DOM lehetővé teszi a dokumentumok írását és olvasását, illetve teljesen új dokumentumok létrehozására is használható. A W3C által megfogalmazott DOM tulajdonképpen nyelvfüggetlen felületek gyűjteménye. A DOM megvalósítása egy adott programozási nyelven egy sor nyelvi kötöttséget von maga után; ezek határozzák meg, hogyan lehet az említett felületeket az adott nyelven használni. Például Javában szabványos módon érhetjük el az objektumok tulajdonságait, míg a Perlnek erre nincsenek eszközei. Ebből következik, hogy Javában a szokásos módon érhetjük el a tulajdonságokat, míg Perlben elemfüggvényeken keresztül kapjuk meg és módosíthatjuk őket.
A jelenleg biztonsággal használható W3C DOM-ajánlás, az úgynevezett DOM Level 1 (első szint). A most bemutatásra kerülő Perl DOM-megvalósítás – ***Enno Derksen*** `XML::DOM` modulja – csak az első szintet támogatja, miként a legtöbb kereskedelmi termék is. Az `XML::DOM` tartalmaz néhány, a W3C-ajánlásban nem szereplő lehetőséget is (terjedelmi okokból azonban most nem mutatom be őket). Megjelent a második szint is, amely már szűrőket is tartalmaz és kényelmesebb bejárást tesz lehetővé, ám az ezt megvalósító értelmezők még fejlesztés alatt állnak. Ha bővebb útmutatásra lenne szükséged az `XML::GDOME` modulról, a ➲ http://tjmather.com/xml-gdome/ oldalon megtalálod.

Csomópontok, csomópontlisták, nevesített csomóponttérképek

A DOM-ban minden objektum három osztályból származtatható: `Node` (csomópont), `NodeList` (csomópontlista) és `NamedNodeMap` (nevesített csomóponttérkép). Egy ***csomópontlista*** egy ***csomópontok***-ból álló tömbnek fogható fel, azaz egy ***csomópontok***-ból álló rendezett listának, amelyben az elemekre számokkal hivatkozhatunk. Például ***csomópontlist***-ád lehet egy adott elem gyermekeiből. A ***nevesített csomóponttérkép*** pedig az asszociatív tömbhöz hasonlítható, vagyis nem más, mint egy ***csomópontok***-ból álló rendezetlen lista, amelyben az elemekre névvel hivatkozhatunk, például ***nevesített csomóponttérkép***-ed lehet egy adott elem tulajdonságaiból (attributum).
A dokumentum minden egyes alkotóeleme egy csomópont-osztályból származó objektumként jelenik meg. A `Document Node` (dokumentum-csomópont) jelenti az egész dokumentumot. Az `Element Node` (elemcsomópont) egy elemet jelképez, egy `Attr Node` (tulajdonság-csomópont) szimbolizál egy tulajdonságot, és a `Text Node` (szövegcsomópont) jelképez egy karakterláncot. Léteznek ***csomópont***-ok olyan dolgokra is, mint a megjegyzések vagy a feldolgozói utasítások. A DOM nagyon jól kihasználja az objektumközpontú programozás lehetőségeit; például a ***csomópont*** összes elemfüggvénye használható egy elemobjektumon, és egy ***csomópontlista*** elem- és szövegobjektumokat is tartalmazhat. Ha egy kicsit összezavarnak az objektumközpontú programozás fogalmai, érdemes egy pillantást vetni a perlboot(1) súgóoldalra.
Egy valóságos példa remélhetőleg átláthatóbbá teszi eddig esetleg kissé homályos benyomásaidat a DOM-ról. Nézzük példadokumentumunkat, melyet az előző két részben is használtunk (az 54. CD Magazin/XML_Perl könyvtárában található). Az utóbbinak a DOM-felépítése egy ***dokumentum-csomópont***-ból fog állni, amelynek egyetlen gyermeke a `dokumentum` nevű ***elemcsomópont***. Amennyiben példadokumentumunk a `<dokumentum>` elemen kívül megjegyzéseket vagy feldolgozói utasításokat tartalmazna, a ***dokumentum-csomópont***-nak lennének ***megjegyzéscsomópont***-, illetve ***feldolgozói utasítás csomópont***-gyermekei. Mivel egy jól formázott XML-dokumentumnak csak egy gyökéreleme lehet, a ***dokumentum-csomópont***-nak elvileg csak egy ***elemcsomópont***-gyermeke lehet. Vegyük észre, hogy a ***dokumentum-csomópont*** dokumentum mint állomány és a `<dokumentum>` mint elem kifejezések nem fedik egymást.
A gyökérelemnek, amely a ***dokumentum-csomópont*** egyetlen ***elemcsomópont***-ja, két gyermeke van. Mind a kettő ***elemcsomópont***, és mindkettő neve `fejezet`. Ezeknek az ***elemcsomópont***-oknak az első tulajdonságai között foglal helyet egy ***tulajdonság-csomópont***, ennek a neve `cim`. Az utóbbinak létezik egy ***szövegcsomópont***-gyermeke, amelynek az értéke `elso`, Továbbá egy `bekezdes` és két másik fejezet ***elemcsomópont***-ja is van.

A fa bejárása

Minden ***csomópont*** objektumnak vannak elemfüggvényei, amelyek segítségével ***csomópont***-ról ***csomópont***-ra bejárhatod a DOM fáját. Egy ***csomópont***-ból megtudhatod, melyek a gyermekei, a szülői és a testvérei. A dokumentum bármely határozott típusáról kaphatsz egy ***csomópontlistá***-t, legyen az a címke neve vagy bármi más. Ez azt jelenti, hogy a DOM segítségével nem kell szigorúan felülről lefelé vizsgálni a dokumentumot, mint az `XML::Parser` fa (Tree) stílusa esetén.
Mint láthatod, a DOM hihetetlenül rugalmas és erőteljes felületet kínál az XML-dokumentumokhoz. A beépített eljárások

```
#!/usr/bin/perl -w

use strict;

use XML::DOM;
use Text::Wrap;

my ($indlevel,@sectnums);

die "Hasznalat: ".$0." {file}\n" unless @ARGV
    ➥== 1;
my $parser = new XML::DOM::Parser;
my $doc = $parser->parsefile($ARGV[0]) or
    die "Nem tudom ertelmezni a
        ➥dokumentumot.\n";

my $root = $doc->getFirstChild;
die "<dokumentum> nem a gyoker elem\n" unless
    $root->getTagName eq "dokumentum";

$indlevel = -1;
$sectnums[0] = 0;
for my $pass (0..1) {
    my $nl = $root->getChildNodes;
    for my $i (0..$nl->getLength-1) {
      my $p = $nl->item($i);
      if ($p->getNodeType == TEXT_NODE) {
          warn "itt nem lehet szoveg" unless
            $pass or $p->getData =~ /^\s*$/;
          next;
      }
      warn $p->getNodeName." nem lehet a
      ➥<dokumentum>-on kivul" unless
          $pass or ($p->getNodeType=
          ➥=ELEMENT_NODE
          ➥and $p->getTagName eq "fejezet");
      process_sect($p,$pass);
    }
}

###############1. szakasz###############
sub process_sect {
    my ($sectnode,$pass) = @_;
    my ($href,$ind);
    if ($pass == 0) {
      ++$sectnums[++$indlevel];
      $sectnums[$indlevel+1] = 0;
      $href =
      ➥join('.',@sectnums[0..$indlevel]);
      $ind = $indlevel;
      $sectnode->setAttribute('href',$href);
      $sectnode->setAttribute('ind',$ind);
      print ' ' x (4*$ind),$href,'
      ➥',$sectnode->getAttribute('cim'),"\n";

###############2. szakasz###############
    } else {
      $href = $sectnode->getAttribute('href');
      $ind = $sectnode->getAttribute('ind');
      print ' ' x (4*$ind),$href,
      ➥'',$sectnode->getAttribute('cim'),"\n\n";
    }
    my $children = $sectnode->getChildNodes;
    for my $i (0..$children->getLength-1) {
      my $p = $children -> item ($i);
      my $t = $p->getNodeType;
      if ($t == TEXT_NODE) {
          warn "szoveg nem lehet a
              ➥<bekezdes>-en kivul" unless
            $pass or $p->getData =~ /^\s*$/;
          next;
      }
      warn $p->getNodeName." nem lehet
      ➥a <fejezet>-ben" unless
          $t == ELEMENT_NODE;
      if ($p->getTagName eq "fejezet") {
          process_sect($p,$pass);
      } elsif ($p->getTagName eq "bekezdes")
{
          my $p1=$p->getFirstChild;
          if ($p1->getNodeType!=TEXT_NODE) {
            warn $p1->getNodeName." nem
            ➥lehet a <bekezdes>-ben" unless
                $pass;
          } else {
            if ($pass) {
                $_ = $p1->getData;
                tr/\n/ /;
                s/^\s+//;
                s/\s+$//;
                my $indent = ' ' x (4*$ind);
                print
                ➥wrap($indent,$indent,$_),
                ➥"\n\n";
            }
          }
      } else {
          warn '<',$p->getTagName,'>
          ➥nem lehet a <bekezdes>-ben';
      }
    }
    --$indlevel;
}
```

megszabadítanak az állapottartó vagy könyvjelző eljárások írásától, amely könnyen felmerülhet egy folyamalapú, de még egy egyszerűbb faalapú értelmező esetén is. Miért akarna bárki is mást használni a DOM-on kívül az XML-értelmezéshez? Talán a redmondi fiúk mégis megtalálták a minden értelemben tökéletes megoldást?

Az igazság az, hogy a DOM elképesztően nehézsúlyú megoldás XML-tartalmak eléréséhez. Egy XML-dokumentum DOM-fája a dokumentum méretének többszörösét falja fel a memóriából. Egy DOM-fa bejárása nem valami gyors és nagyon sok függvényhívással jár, ami Perlben eléggé erőforrás-igényes. Teljesítmény szempontjából határozottan rossz megoldás CGI esetén DOM-ban gondolkozni, ugyanis használata elfogadhatatlan válaszidőkkel és nagy terheléssel járna a kiszolgálón. Mindazonáltal a DOM alkalmazása gyakran leegyszerűsítheti a programodat, javíthatja az átláthatóságot. Tegyük fel, hogy

van egy alkalmazásod, amelyben a sebesség fontos ugyan, de a dokumentum elhanyagolhatóan ritkán változik a program futtatásának számához képest. Elgondolkodtató lehetőség a kinyert adatok tárolása valamilyen egyszerűbb formában, mondjuk DBM adatbázisban. Az alkalmazásod ekkor az utolsó módosítás dátumát összehasonlíthatja az eredeti XML-dokumentumra és az abból nyert adatbázisra vonatkozóan, és csak akkor szükséges újra értelmezni a dokumentumot, ha a lényegi adatokat tartalmazó adatbázis már nem naprakész. Szokásos esetben csak betölti az adatbázist. Amíg nem dolgozol nagyon nagy mennyiségű adattal, addig ez a módszer sokat javíthat az átlagos futási sebességen.

Eleget beszéltünk, lássuk a kódot!

Ahogy azt megtippelhetted, múlt havi programunkat fogjuk újraírni, az `XML::Parser` fastílusa helyett az `XML::DOM` modult használva. Elég nagyvonalúan kezeljük majd a hibakezelést a dolgok egyszerűsítése érdekében. Lássuk!

```
#!/usr/bin/perl -w

use strict;

use XML::DOM;
use Text::Wrap;

my ($indlevel,@sectnums);

die "Hasznalat: ".$0." {file}\n"
➥unless @ARGV == 1;
my $parser = new XML::DOM::Parser;
my $doc = $parser->parsefile($ARGV[0]) or
     die "Nem tudom ertelmezni a
          ➥dokumentumot.\n";
```

Mivel az `XML::DOM` az `XML::Parser`-ből származik, ugyanazokat a `parse()` és `parsefile()` eljárásokat használjuk. Ha az értelmezés sikeres volt, a visszatérési érték egy hivatkozás a ***dokumentum-csomópont***-ra.

```
my $root = $doc->getFirstChild;
die "<dokumentum> nem a gyoker elem\n" unless
     $root->getTagName eq "dokumentum";
```

Azt akarjuk, hogy a ***dokumentum-csomópont***-unknak csak egy gyereke legyen, egy ***elemcsomópont*** dokumentum névvel.

```
$indlevel = -1;
$sectnums[0] = 0;
for my $pass (0..1) {
     my $nl = $root->getChildNodes;
     for my $i (0..$nl->getLength-1) {
       my $p = $nl->item($i);
```

Kapunk egy ***csomópontlistá***-t, amely gyökérelemünk gyermekeit tartalmazza. A ***csomópont***-ok a listában nullától vannak számozva. Az egyes gyermekeket a ***csomópontlista*** `item()` elemfüggvényével érjük el. A `getLength` mondja meg, hogy hány gyermek található a listában.

```
   if ($p->getNodeType == TEXT_NODE) {
        warn "itt nem lehet szoveg" unless
        $pass or $p->getData =~ /^\s*$/;
        next;
   }
```

A `getData()` elemfüggvény a ***szövegcsomópont***-ok sajátja. Más nyelvekben esetleg `getNodeValue()`-ként fordul elő. Lásd az `XML::DOM` POD-ját.

```
    warn $p->getNodeName."
    ➥nem lehet a <dokumentum>-on kivul" unless
         $pass or ($p->getNodeType==ELEMENT_NODE
         ➥and $p->getTagName eq "fejezet");
    process_sect($p,$pass);
     }
}
```

A `fejezet` elemek feldolgozását a `process_sect()` függvényre bízzuk. (Lásd a lista 1. szakaszát.)
Itt az ***elemcsomópont*** `getAttribute()` és `setAttribute()` elemfüggvényeit használtuk. Ezek karakterláncokat várnak el és adnak vissza. A ***tulajdonság-csomópont*** `getAttributeNode()` és `setAttributeNode()` függvényeivel könnyű őket összekeverni, úgyhogy különös figyelemmel járj el a használatukkor. (Lásd a lista 2. szakaszát.)
A `getFirstChild()` pontosan azt teszi, amit a neve is sugall. Itt mi vakon bízunk abban, hogy `<bekezdes>` elemünk egyetlen gyermeke egy ***szövegcsomópont***.

```
       if ($p1->getNodeType!=TEXT_NODE) {
     warn $p1->getNodeName." nem lehet a
<bekezdes>-ben" unless
          $pass;
       } else {
     if ($pass) {
          $_ = $p1->getData;
          tr/\n/ /;
          s/^\s+//;
          s/\s+$//;
          my $indent = ' ' x (4*$ind);
          print wrap($indent,$indent,$_),"\n\n";
     }
       }
   } else {
       warn '<',$p->getTagName,'>
                ➥nem lehet a <bekezdes>-ben';
   }
     }
     --$indlevel;
}
```

Végszó

Remélem, sok újat és hasznosat sikerült megmutatnom ebben a négyrészes sorozatban. Érdemes legalább egyszer kipróbálni az XML-t, mert sok feladatot általánosítani lehet vele. Ugyanakkor tudni kell, hogy hol lehet használni és éppen melyik értelmezőt, mert a segítségével könnyen írhatunk memóriafalót is. Tanuld meg és használd, csak nyerhetsz vele.

Fülöp Balázs (admin@guardware.com)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mrożek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.