Sablon alapú XML-értelmezés C++ alatt

A Xerces könyvtár és egy kis C++-kód segítségével könnyen kezelhető objektumokba gyűjthetjük az XML-fájlokból az értelmezni kívánt adatokat.

Az XML kulcsszó alapú adatleíró nyelv, amit arra terveztek, hogy adatainkat beszédes, egyedileg választott kulcsszavakkal rendszerezhessük. Az XML feladata, hogy elkülönítse az adatot és annak felhasználását, illetve gép- és kiépítésfüggetlen módon tegye lehetővé az adatok mozgatását különböző alkalmazások között. Az XML egy másik hasznos alkalmazási lehetősége, amikor a folyamatokat logikus és átlátható módon írjuk le, és az alkalmazás futásidőben is végrehajthatja őket.

XML értelmezése

Ahhoz, hogy az XML-állományt sikeresen elemezhesse, a fejlesztőnek előbb létre kell hoznia egy olyan fájlt, amelyet az értelmező feldolgozhat. Az értelmező az XML-fájl beolvasását és értelmezését végző osztott objektumok készlete.
Kétfajta értelmező létezik: érvényesítő és nem érvényesítő. Az érvényesítő értelmező végignézi az XML-állományt és megállapítja, hogy az jól formált-e, megfelel-e a megadott XML-sémának vagy dokumentum-típusmeghatározásnak (DTD). A nem érvényesítő értelmezők egyszerűen beolvassák a fájl, és figyelmen kívül hagyják az XML-sémában vagy DTD-ben megadott szerkezetet.
Az általánosan használt értelmezők két eltérő megközelítést alkalmaznak: az eseményvezérelt és a faalapú megoldást. Az eseményvezérelt értelmezőt SAX-nak nevezik (egyszerű API az XML-feldolgozáshoz, angolul simple API for XML processing). A faalapú értelmező az XML-fájl beolvasása és értelmezése közben a memóriában egy DOM (document object model) fát hoz létre.
A DOM-megközelítésnél nehezebb eligazodni, és nem teszi lehetővé az XML-elemek és a tartományokhoz tartozó objektumok közötti egyszerű összerendelést. A SAX-események alkalmazásával lehetővé teszi a fejlesztők számára, hogy az XML-fájl beolvasása és értelmezése közben hozzák létre saját tartományra jellemző objektumait. Ebben a cikkben egy, az XML-értelmezéshez SAX API felületet használó keretrendszert ismerhetünk meg.

XML-értelmezők a C++ nyelvhez

A két legáltalánosabban használt C++ alapú értelmező az Apache projekt nyílt forrású Xerces rendszere és az IBM alphaWorks projektje keretében létrejött XML4C. Az XML4C a Xercesen alapul.
Mindkét értelmező lényegében ugyanazt a forrás- és könyvtárfelületet nyújtja, így felcserélhetők egymással. Egyaránt támogatják a DOM és a SAX alapú XML-értelmezést.
Az írásunkban bemutatott megoldás a Xerces értelmezőt és a SAX-megközelítést használja. Az XML-értelmezéshez kapcsolódó Xerces-források vagy bináris állományok a Xerces-weblapról tölthetők le (lásd a ***Kapcsolódó címeket***).

XML-fájlok értelmezése SAX-szal

Ahhoz, hogy az XML-állományokat elkezdhessük a SAX API segítségével értelmezni, előbb meg kell értenünk a SAX C++ objektumkapcsolatokat. A SAX-hoz két alapvető felületet terveztek (*1–2. táblázat*).

1. táblázat

SAXParser
setDoValidation
setDoNamespace
setDoSchema
setValidationFullSchemaChecking
setDocumentHandler
setErrorHandler
parse

2. táblázat

HandlerBase
warning
error
fatalError
startElement
characters
ignorableWhitespace
endElement

Tüzetesen megvizsgálva a `HandlerBase` objektumot, alapvetően két fajta tagfüggvényt figyelhetünk meg: vannak hibakezelő és dokumentumfeldolgozó tagfüggvények. A hibakezelő tagfüggvények közé tartozik a `warning`, az `error` és a `fatalError`, az értelmező tagfüggvények pedig a `startElement`, `characters`, `ignorableWhitespace` és az `endElement` lesznek. Mint később látni fogjuk, ezeket a viselkedési módokat külön objektumokra választhatjuk szét.
A `SAXParser` osztály felügyeli az alaptulajdonságok beállítását és a futásidőben érvényre juttatott, kívánt viselkedést.
A következő példakód bemutatja az XML-fájlok értelmezésének alaplépéseit, SAX értelmezővel C++ nyelven:

```
// SAX értelmező új példányának létrehozása
SAXParser parser;

// kívánt viselkedés alaphelyzetbe állítása
parser.setDoValidation(true);
parser.setDoNamespaces(true);
parser.setDoSchema(true);
parser.setValidationSchemaFullChecking(true);

// kezelők felvétele a dokumentumhoz és
// hibakezeléshez
parser.setDocumentHandler(&docHandler);
parser.setErrorHandler(&errorHandler);

// Fájl értelmezése
parser.parse("MyXMLFile.xml");
```

Az értelmezés pillanatában az általunk létrehozott osztályok (a `docHandler` és az `errorHandler`) továbbítódnak az értelmezés során kiváltott eseményhez. Ezek az osztályok a `HandlerBase` `Xerces` alaposztályból származnak, és felülírják a megfelelő tagfüggvényt, hogy a saját feladatkategóriájukhoz tartozó eseményeket kezelni tudják.
Most, hogy bemutattuk az XML SAX alapú értelmezését,

nézzük meg, hogyan készül el az XML-keretrendszer, és miképpen használja ki az API-ban rejlő lehetőségeket.

Rendszabályosztályok

Andrei Alexandrescu meghatározása a Modern C++ Design című művében (lásd a ***Kapcsolódó címeket***) ismertetett és híressé vált rendszabályosztályokra (policy class) a következő: „osztály-csatolófelületeket vagy osztálysablon-felületeket adnak meg. A csatolófelület egy vagy az összes elemet tartalmazza a következők közül: belső típusmeghatározások, tagfüggvények és tagváltozók."

A rendszabályosztályok igazi hasznát az XML-keretrendszerben akkor látjuk majd, amikor sablonalapú C++-rendszerben hozzuk létre őket. A rendszabályok segítségével egészen finom felbontással paraméterezhetjük és állíthatjuk be a képességeket. Ebben a felállásban a rendszabályok a következő feladatokat támogatják: dokumentumkezelés, hibakezelés, tartománykiosztás (domain mapping) és értelmezés (parsing).

Ha ezeket az elemeket rendszabályokként hozzuk létre, sokkal tömörebb kódot tudunk készíteni, amit egy C++ nyelvben és a sablonhasználatban jártas programozó könnyebben karbantarthat.

A XML-értelmező keretrendszer elsődleges osztálya az `XMLSAXParser`. Ez a testreszabhatóra tervezett osztálysablon tagváltozóként tartalmazza az `XMLParserInterface`-t és a `SAXParser` objektumot. A dokumentum- és a hibakezelők rendszabályosztályai egyaránt a sablonértékekhez tartoznak. Miután a megfelelő kezelőket és más tulajdonságokat beállítottuk, végső soron minden értelmezés a `SAXParser` tagváltozóhoz kerül.

Saját kezelőinket a keretrendszerrel szinte magától értetődő módon tudjuk rendszabályosztályként beilleszteni. Az ilyen típusú felépítésnek nagy előnye, hogy egy vagy több rendszabály megváltoztatásával ugyanazt a keretrendszert használhatjuk a különféle értelmező API-kban és eltérő tartománykiosztás-objektumokhoz – ezt a gyakorlatot azonban nem mutatjuk be cikkünkben.

Saját kezelők létrehozásához vezessünk le újonnan készített saját osztályokat a `HandlerBase`-ből, és írjuk felül a minket érdeklő virtuális tagfüggvényeket. A következő két saját kezelőtípus az `XMLFactory` keretrendszerben készült (*3–4. táblázat*).

3. táblázat

XMLSAXHandler

startElement
character
ignorableWhitespace
endElement

4. táblázat

XMLSAXHandler

warning
error
fatalError

Az `XMLSAXHandler` vezérli a dokumentumok eseményfeldolgozását, a `XMLSAXErrorHandler` a különböző hibavisszahívásokat kezeli.

XML-tagok és tartományobjektumok összerendelése

XML-értelmező keretrendszerünk következő feladata az XML-tagok átalakítása tartományfüggő objektumokká, amelyet azután az alkalmazásban sablonok és laza rendszabályosztály-meghatározások segítségével fel tudunk használni.

Az `XMLDomainMap` sablon egyetlen, `XMLNode` nevű sablonértéket fogad el. Tartomány-hozzárendelő objektumunk felülete a következőképpen néz ki (*5. táblázat*).

5. táblázat

XMLDomainMap

create
add
updateAttribute

Az `XMLNode` a gyermekeket alfákban egyesítő faszerkezetben egyszerre tölti be a gyökér és a levelek szerepét. Az `XMLNode` csatolófelület a következőképpen néz ki (*6. táblázat*).

6. táblázat

XMLNode

operator==
operator!=
operator=
addChild
hasChildren
numChildren
value
name
getChildCount
getChild
getParent

Az egész módszer kulcsa az objektum nyilvános felületének megtervezése. Megfigyelhetünk néhány műveleti-felülírást is, ilyen például az egyenlőségjel (operator==), a nem egyenlő (operator!=) és az összerendelés műveleti jel (operator=). Ezeknek az az előnye, hogy az objektumot mostantól rengeteg szabványos sablonkönyvtárral (STL), tárolóval és algoritmussal használni tudjuk, kiaknázva a C++ nyelv különleges képességeit.

Osztályok összefűzése – az XML Façade

Eddig különálló osztályokra és az XML-értelmező keretrendszerünkhöz készített sablonok ismertetésére összpontosítottunk. A következő lépésben összekapcsoljuk a különálló felületeket, így azok a façade tervezési mintának megfelelően a külvilág számára egyetlen, összefüggő egységnek látszanak majd.

7. táblázat

XMLProcessor

parse
getParseEngine

A façade tervezés egyszerű és elegáns módja annak, hogy a külső ügyfélről az értelmezéshez használt belső rendszabály osztályokba vigyük át az értelmezőszolgáltatást. A tervezési mintákban (Design Patterns) a szerzők a következőképpen foglalták össze céljaikat: „Célunk az alrendszer csatolófelület-készletéhez egységes felületet készíteni. A façade olyan magasabb szintű felületet határoz meg, amelynek segítségével az alrendszer könnyebben használhatóvá válik." A létrehozott façade neve XMLProcessor. Ezt a következő felülettel határozták meg (*7. táblázat*).

Miután minden forrást beírtunk, a példaprogram futtatásához szükség lesz egy XML-fájlra, illetve egy próbaügyfélre.

Az XML-fájl értelmezése

A keretrendszer egyszerű működésének bemutatására egy egyszerű (nevet és számlaszámot tartalmazó) vásárlóadatlapot leíró XML-állományt készítettünk:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<customer>
     <name>John Doe</name>
     <account-number>555123</account-number>
</customer>
```

Készítsük el ezt az állományt valamilyen szövegszerkesztővel, és mentsük *MyXMLFile.xml* néven.

Nyilvános felület – ügyfélalkalmazás elkészítése

Az ügyfélalkalmazáshoz szánt tömör csatolófelületet a keretrendszer segítségével fogjuk elkészíteni.
Az ügyfélkeretrendszer által használható elsődleges tagfüggvények a következő, egészen rövid példa C++-forráskóddal írható le:

```
// ---------------------------------------
//  Példaforráskód az XML doc értelmezéséhez
// ---------------------------------------
#include "XMLProcessor.hpp"
#include "XMLDomainMap.hpp"
#include "XMLSAXParser.hpp"
#include "XMLNode.hpp"
#include "XMLCommand.h"
#include "XMLSAXHandler.hpp"
#include "XMLSAXErrorHandler.hpp"

#include <iostream>
using namespace std;
using namespace XML;

// Először is tegyük félre a ronda dolgokat
typedef XMLSAXHandler<XMLDomainMap<XMLNode> >
➥DOCHANDLER;
typedef XMLSAXErrorHandler ERRORHANDLER;
typedef XMLSAXParser<DOCHANDLER, ERRORHANDLER>
➥PARSER;
typedef XMLProcessor<PARSER> XMLEngine;

// alap-tesztügyfél létrehozása
int main(void)
{
     // a fájlnevünket hordozó
     // karakterlánc-objektum létrehozása
     std::string xmlFile = "MyXMLFile.xml";

     // Az XMLFactory egy példányának
     // létrehozása
     XMLEngine parser(xmlFile);

     //  rvényesítés kikapcsolása
     parser.doValidation(false);

     // XML fájl értelmezése
     parser.parse();

     // Fa gyökerének lekérése
     XMLNode root = parser.getXMLRoot();

     // Objektumunk nevének kiírása
     cout << "Root = " << root.name() << endl;
     return 0;
}
```

A fa gyökerének megfelelő `XMLNode` objektumpéldányunk értelmezése után a gyökér `XMLNode` gyermekei is elérhetővé válnak.

Példaügyfél fordítása

Az utolsó lépés az ügyfél fordítása lesz. A fordítást egyszerűen parancssorban végezzük:

```
g++ -o testClient -I.
➥-I/path/to/xerces/include
➥-I/path/to/xerces/include/xerces
➥testClient.cpp -L/path/to/xerces/lib
➥-lxerces-c
```

A fenti sor lefordítja nekünk az ügyfélalkalmazást. Ezután a teszt futtatása következik. Ne feledjük el úgy beállítani a `LD_LIBRARY_PATH` környezeti változót, hogy a helyi Xerces-telepítésünk ***lib*** könyvtárára mutasson. Mivel az osztott programkönyvtárak ebből a könyvtárból származnak, a hibátlan működés érdekében az alkalmazásbetöltőnek futásidőben valamilyen módon el kell tudnia érni a szükséges szimbólumokat.

A `testClient` futtatásakor a következőknek kell megjelenniük:

```
$>testClient

Adding child name
Adding child account-number
Root = customer
```

Mostantól egy teljes mértékben működőképes, C++-sablonokon alapuló XML-értelmező keretrendszerrel rendelkezünk, amelynek segítségével az XML technológiát beépíthetjük új, vagy már meglévő programjainkba. A példakód megtalálható a 52. CD Magazin/XML könyvtárában, illetve letölthető az ➲ ftp.ssc.com/pub/lj/listings/issue110/6655.tgz címről.

Linux Journal 2003. június, 110. szám

John Dubchak
Vezető programfejlesztő, aki tanácsadóként dolgozik Saint Louis környékén. Az elmúlt 12 évben C++ nyelven fejlesztett, és el sem hiszi, hogy az első C++ kódsorai mennyire rosszak voltak.

KAPCSOLÓDÓ CÍMEK

Xerces ➲ http://xml.apache.org/xerces-c
Xerces 2.1.0 (a cikkben ezt a változatot használtuk)
➲ http://xml.apache.org/dist/xerces-c/stable/xerces-c-src2_1_0.tar.gz
Andrei Alexandrescu Modern C++ Design, Addison Wesley Kiadó, 2002. 7–8. oldal
Erich Gamma, *Richard Helm*, *Ralph Johnson* és *John Vlissides* Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Kiadó, 1994.