Adatkezelés XML-es környezetben 2012-es kérdéssor kidolgozás

by Huzynets Erik

Kérdések

1. XML kialakulása, története

1967 Tunnicliffe tanulmánya, 1969 GML nyelv, 1983 SGML nyelv, 1993 HTML nyelv. Első fő hivatalos dokumentuma a 98-ban megjelent XML 1.0 W3C ajánlás. Ebben az XML-t az SGML jelölőnyelv részeként kezelik. Megjelenésében fő szerepet játszott a hatékonyság növelése, és egyesek szerint az SGML-től való látványos elkülönülés.

2. XML jellemzése, előnyei, szerepe

Jellemzői : az adatok és metaadatok együtt vannak tárolva, a formátuma szöveges, tetszőleges tartalom tárolható benne, a szerkezete rugalmas, a kezelő felületi szabványok gazdag készletével rendelkezik.

Előnyei: platformfüggetlen, hatékonyan alkalmazható az interneten, egyszerű szerkezetű, felhasználó által olvasható formátumú, könnyen implementálható, XML dokumentum feldolgozó szabványok hada létezik.

Szerepe: megalkotásának célja az volt, hogy legyen egy nyelv, ami megőrzi az SGML rugalmasságát, de elég egyszerű a hatékony implementációhoz. Használható adatleírásra, tárolásra, megjelenítésre, adatcserére.

3. XML megjelenési alakjai:

- primér (a nyers szöveg a szövegszerkesztőben)
- szekundér (a böngészőben megjelenő, feldolgozott szöveg)
- infoset (a dokumentumot reprezentáló teljes fa)

4. XDM modell jellemzése, csomóponttípusok:

XDM (Xquery and Xpath Data Model).
Az XDM alapvetően az Infoset modellre épül, de kiegészíti azt az alábbi elemekkel:
• típuskezelés, az XMLSchema adattípusainak bevonása
• összetett szerkezeti struktúrák megjelenése
• szekvencia bevezetése
• elemek rendezettségének ?gyelése a szekvencián belül

csomóponttípusok:
• dokumentum egység (az XML fa gyökere)
• jelölőelem egység (element)
• elemjellemző (attribute)
• feldolgozási utasítás (processing instruction)
• egyedhivatkozás (entity reference)
• formátumjelölő (notation)
• karakter egység (character)
• megjegyzés (comment)
• DTD egység (dokumentum séma szerkezete)
• névtér egység (namespace)
• nem-elemzendő egyed (unparsed entity)

5. XML helyesen formáltsága

Az XML dokumentum szöveges állományban tárolt, szokásos kiterjesztése "xml". A jelölő elemek használatosak a leíró információk, metaadatok megadására. Az elemek tagjait a '<' és '>' karakterek határolják. A tartalom elemek lehetnek egy- vagy kéttagúak, nevük tetszőleges szó lehet.

A jellemzők az elem valamely tulajdonságát, viselkedési paraméterét adják meg. <tag .. jellemző_neve="érték" ..> . Egy elemhez több jellemző is rendelhető, értékük elemi. Az elemhez szorosan kötődő értéket tárolnak.

PI alakja: <?kezelő .. ?> .

Megjegyzés alakja: <!-- ... --> .

Az XML dokumentum első sorának egy XML deklarációs elemet kell tartalmaznia (feldolgozói utasítás, PI): <?xml version="verzio" ?> (jelenleg 1.0 van). Minden elemnek csak egy szülője lehet - hierarchia. A dokumentum gyökér eleme az a tartalom elem, amelyhez nem létezik szülő elem.

6. XML elem típusok, attribútum

Tartalom elem: a felhasználó által megadott tartalom tulajdonságait írja le, a feldolgozó programnak szóló információkat tartalmaz. Lehet egy- vagy kéttagú. <életkor> 23 </életkor>

Deklarációs elem: a feldolgozónak szóló instrukciókat tartalmaz. <?elemnév jellemzők_listája ?>

Megjegyzés elem: az olvasónak, programozónak szóló információkat tartalmaz. <!-- megjegyzés -->

Attribútum: az elem valamely tulajdonságát, viselkedési paraméterét adja meg. Megadása egy név - érték párossal történik. jellemző_neve = "érték" A jellemzőhöz kapcsolt értéket mindig idézőjelek között kell megadni.

7. PI elemek jellemzése, speciális karakterek kezelése, megjegyzések

A PI egy feldolgozói utasítás, ami az XML feldolgozó programnak szól. Az XML-ben is vannak foglalt karakterek, amik nem lehetnek normál szöveg részei. Ezek használatára egyed-szimbólumokat használnak (&kod; formátum):

< : < > : > & : & ' : ' " : " .

A számkóddal azonosított karakterek esetén a szimbólum megadása : &#nnnn; ahol nnnn az igényelt karakterhez tartozó Unicode érték.

Összefüggő szövegrész beillesztése szeparátor elemek keresése nélkül a CDATA elemmel történik: <![CDATA[szöveg]]> .

8. XML normalizált alak, validáltság, jól formáltság

Az XML dokumentumot normalizáltnak nevezzük, ha minden szóköz-ekvivalens karaktersorozat csak egy karakter hosszú.

Egy XML dokumentumot validáltnak, ellenőrzöttnek nevezünk, ha az teljesíti a megadott DTD (vagy más sémaleíró) követelményeit.

9. Névtér szerepe, használat módja

A névtér egy tématerületet azonosít. Az elemeket társítani lehet a névterekkel, minden elemehez egy névtér rendelhető. A névtér neve maga is tetszőleges. A névütközés esélyének csökkentésére a névtér megadásakor URL formátumú azonosítót használnak. Ennek a szerver része a fejlesztő saját szerverének elérési neve, specifikus része rendszerint a feldolgozó program tárgykörére utal.

pl. http://www.w3.org/2000/10/XMLSchema : W3C XMLSchema névtere.

A névterek megadása elemjellemzőkön keresztül történik. A vonatkozó elemjellemző az 'xmlns' .
Megadás:

<elemnév ... xmlns:alias = URL ...>

10. Névtér alias megadása, hatásköre

A terebélyes URL formátumú névtér azonosítók helyett lehetőség van lokális alias név használatára. Megadás: <elemnév ... xmlns:alias = URL ...> ahol a megadott alias használható a továbbiakban a kapcsolódó URL helyett. Az aliasnév érvényességi köre azon elemre és annak befoglalt elemeire terjed ki, amelyben definiálták. Egy belső elemben a kívül értelmezett aliasnév felülírható egy új értelmezéssel. A névtér hozzárendelése egy elemhez: <alias:elemnév ...> . Jellemzőhöz: <elemnév alias:jellemző_név='érték' ...> .

11. Default névtér használata

Az xml szabvány lehetővé teszi, hogy default névteret hozzunk létre, amikor nem kell a névtér aliast sem megadni. <elemnév ... xmlns = URL ...> . Ha egy elemhez akarunk névteret hozzárendelni, akkor a normál <elemnév ...> megadással tehetjük ezt meg. A default névtér csak az elemekre vonatkozik, a kapcsolódó elemjellemzőkre már nem.

12. XML feldolgozási módok:

-SAX api
SAX = Simple Api for XML (önkéntes kezdeményezés)
Az értelmező szekvenciálisan dolgozza fel az XML dokumentumot. Minden fontosabb eseményről értesíti a kezelő programot a callback mechanizmussal.
-DOM api
DOM = Document Object Model (objektum orientált)
Az értelmező beolvassa az XML dokumentumot és egy fát, mint objektumot épít belőle. Az előállított objektum-fát a metódusain keresztül lehet olvasni és módosítani.
-xslt
XSLT: XML Stylesheet Transformation Language
Dokumentum-fa modell (de nem DOM). Egyszerűbb, nem módosítható.
-XQery
Az XQuery célja egy imperatív lekérdező nyelv biztosítása.

13. DTD szerepe, jellemzése

A DTD szerepe a dokumentumok szerkezetének korlátozása. Az SGML nyelvből öröklődött. A DTD leírás egy séma leírásnak tekinthető. A DTD nem XML formátumú, egyedi formalizmusa van. Alapelemei: jelölő elemek, elemjellemzők, szimbólumok, egyedek.

14. DTD megadás módjai

A DTD esetében a séma leírása lehet az XML dokumentum elején: <!DOCTYPE gyökérelem_neve [séma_leírás]> ,

vagy külön fájlban: <!DOCTYPE gyökérelem_neve SYSTEM "állomány" >.

15. ELEMENT DTD elem, számosság jelzése

Az egyed szimbólumok megadásakor az ELEMENT-et kell használni. Az elem megadása:
<!ELEMENT elemnév szerkezet> . A szerkezet főbb típusai:

- EMPTY : üres elem, egytagú tartalom elem
- (#PCDATA) : szöveg értéket tartalmazó elem
- ANY : tetszőleges tartalom
- (szerkezet): gyerekelemeket tartalmazó elem (ELEMENT-ONLY típus)
- (#PCDATA | szerkezet)* : vegyes szöveget és gyereket is tartalmazó elem (MIXED típus).

Számosság jelzése:
()* : tetszőleges számú előfordulás
()+ : egy vagy több előfordulás
()? : nulla vagy egy
() : pontosan egy.
Az elemek viszonya:
e1 , e2 : az elemek szekvencája
e1 | e2 : a két elemből az egyik fordul elő.

16. ATTLIST DTD elem

Az elemjellemzők megadására van, alakja:
<!ATTLIST elem jellemző_neve típus alapérték>
Az érték típusa pontosan nem adható meg, csak a jellege: CDATA (szöveges), (e1|e2) (értéklista), ID (egyedi azonosító, kulcs jelleg, csak egy lehet elemenként), IDREF (egy ID értéke, idegen kulcs jelleg), IDREFS (több ID értékei), NMTOKEN (a jellemző XML azonosító nevet tartalmaz), ENTITY (egy egyed értéke), NOTATION (külső objektumra utalás).
Az alapérték rész azt szabályozza, hogy az elemjellemzőnek kötelező-e értékkel rendelkeznie: #REQUIRED (kötelező explicit érték), #FIXED (egy rögzített érték, FIXED után kell megadni), #IMPLIED (nem kötelező), érték (default, ilyenkor IMPLIED-et használja az értelmező).

17. szimbólumok és paraméter szimbólumok használata (DTD):

szimbólumok megadása:
<!ENTITY szimbólum "érték">
<!ENTITY szimbólum SYSTEM "állomány">
Szimbólum felhasználása
< > ... &szimbólum; ... < >

Paraméter-szimbólumok: a definíciós részben használható:
<!ENTITY % szimbólum érték>
<!ENTITY % szimbólum SYSTEM "állomány">
Paraméter felhasználása:
<!DOCTYPE ....%szimbólum; ... >

18. feltételes DTD sémarészek használata, NOTATION szerepe

DTD IGNORE és INCLUDE elemek használata: feltételes értelmezési szekciókat lehet velük megadni.
<![IGNORE[egy bizonyos rész kihagyása
séma definíció
]]>
<![INCLUDE[egy bevonandó rész behatárolása
séma definíció
]]>

DTD NOTATION elem: elemjellemzőhöz, vagy egyedhez kötődik, célja külső objektum (feldolgozó program) azonosítása.
<!NOTATION név SYSTEM feldolgozó>

19.XDM elemek sémáinak megadása DTD-vel

20. elemtípusok megadása DTD-ben

Az egyed szimbólumok megadásakor az ELEMENT-et kell használni. Az elem megadása:

<!ELEMENT elemnév szerkezet> . A szerkezet főbb típusai:

- EMPTY : üres elem, egytagú tartalom elem
- (#PCDATA) : szöveg értéket tartalmazó elem
- ANY : tetszőleges tartalom
- (szerkezet): gyerekelemeket tartalmazó elem (ELEMENT-ONLY típus)
- (#PCDATA | szerkezet)* : vegyesm szöveget és gyereket is tartalmazó elem (MIXED típus).

21. ER konverzió DTD-re:

egyed-elem
elemi tulajdonság->#PCDATA típusú gyerekelem
Kulcs tulajdonság->ID típusú jellemző
összetett tulajdonság-> elemeket tartalmazó gyökérelem
többértékű tulajdonság->gyerekelem,ismétlődéssel
1:N kapcsolat-> IDREF típusú jellemző a gyerekelemnél
N:M kapcsolat-> egy kapcsolóelem létrehozása, melynek van két IDREF típusú jellemzője a kapcsolt elemekre
kötelező kapcsolat-> az IDREF jellemző REQUIRED típusú

22. DTD korlátai:

A DTD szintaktikája nem felel meg az XML szintaktikának, nem lehet megkötést adni a szövegértékekre, nem támogatja a különböző adattípusok kezelését, kevés támogatott integritási feltétel van, nem elég rugalmas a tartalom modell, nem tudja kezelni a névtereket, nem alkalmas a moduláris sémafejlesztésre (nincs öröklés), hiányos az idegenkulcs opció (id, idref korlátozottsága).

23. DTD integritási elemei

ID: egyedi azonosító érték, a kulcs integritási megkötéshez hasonló. Egy elemnél csak egy ilyen típusú jellemző lehet. Az ID olyan kulcsot jelent, amelynek egyedisége nem reláció-, hanem adatbázis hatáskörű.

IDREF: egy ID értékét tartalmazza, hasonlít az idegen kulcs integritási feltételhez. A DTD szemléletében az IDREF típusú értéknek a dokumentum ID értékek halmazából kell kikerülnie. Így nem tudjuk előírni, hogy milyen elemtípusra mutasson a hivatkozás.

24. XMLSchema jellemzése, használata, összevetés a DTD-vel

XMLSchema jellemzése, használat módja: XML szintaktikája van, típusokat használ, gazdag gyári adattípusokkal rendelkezik, egyediek az elemtípusai, nevesítettek a típusai, specializációs kapcsolat van a típusok között, kezeli az absztrakt típusokat, szigorúbbak a kulcs és idegenkulcs megkötései, támogatja a névtereket, gazdag struktúra elemekkel rendelkezik, moduláris a szerkezete.

```
<xs: element name = type="kl:dolg_t">
        <xs:key name=...>
        </xs:key>
</xs:element>
```

25. Szerkezet megadás módjai XMLSchema-ban

összetett szövegelem:element/sympleType/list
üres elem: element/complexType
vegyes elem: complexType(mixed=true)
elemjellemzővel ellátott elem: element/complexType/attribute
számosság jelzése: minOccours,maxOccours(=unbounded: végtelen előfordulás)
kulcs:element/complexType/key/ selector és field xpath
idegen kulcs: element/complexType/keyref/ selector és field xpath

Szekvencia: <xs:sequence>szekvencia</xs:sequence>
opciólista: <xs:choice>elemek</xs:chice>
tetszőleges sorrend: <xs:all>elemek</xs:all>

26. Elem megadás szintaktikája XMLSchema-ban

Elem típusok létrehozása XMLSchemaban:

```
<xs:simpleType name="tipusnev" >      simpletype-nál csak szöveg tartalma lehet az elemnek és nem tartozhat hozzá jellemző
<!-- leiras -->
</xs:simpleType>
```

<xs:complexType name="tipusnev" > szerepelhet "mixed" jellemző, ami ha true akkor szöveg és gyerekelem egyaránt előfordulhat
<!-- leiras -->
</xs:complexType>

27. Gyerekelem struktúrák megadása XMLSchemaban

Komplex típusok esetén lehetőség van a gyermek elemek struktúrájának megadására. A konstrukciós módszerek:
- sequence: szekvenciális megadás, rögzített sorrend
- choice: a megadott elemekből egyet lehet kiválasztani felhasználásra
- all: a megadott elemek maximum egyszer fordulhatnak elő, tetszőleges sorrendben.

Ezek a konstrukciós modulok egymásba ágyazhatóak.

28. Text-only noname típus XMLSchemaban

Csak szöveg: <a>szöveg

29. Empty noname típus XMLSchemaban

Üres: <a>

30. Child-only noname típus XMLSchemaban:

Csak új tag lehet: <a>

31. Mixed noname típus XMLSchemaban:

Szöveg és új tag: <a>szöveg

32. Attributum megadás XMLSchemaban

<attribute name="jellemzo_neve" type="tipus" parameterek />

33. Text-only attributummal noname típus XMLSchemaban: ???

34. Elsődleges kulcs megadása XMLSchema-ban

REF alapú típuskijelölés XMLSchemaban: az elemet definiáló részben szerepelhetnek a tárolt értékeket befolyásoló megkötések, ezeket egy gyerek elemmel írjuk le. Az értelmezett megkötések:

key: elsődleges kulcs
keyref: idegen kulcs
unique: egyediség.

A kulcs megadása:

```
<xs:element name="nev" type="tipus">
        <xs:key name="megkötés neve">
                <xs:selector xpath="elérési_út1">
                <xs:field xpath="elérési_út2">
```

```
        </xs:key>
</xs:element>
```

35. Idegen kulcs megadása XMLSchema-ban

```
<xs:element name="nev" type="tipus">
        <xs:keyref refer="névtér:kulcs neve" name="idegen kulcs neve">
                <xs:selector xpath="eleresi ut" />
                <xs:field xpath="eleresi ut 2" />
        </xs:keyref>
</xs:element>
```

36. Típusok kategóriái és saját típus megadásának módjai xmlschema-ban

összetett szövegelem:element/sympleType/list
üres elem: element/complexType
vegyes elem: complexType(mixed=true)
elemjellemzővel ellátott elem: element/complexType/attribute
számosság jelzése: minOccours,maxOccours(=unbounded: végtelen előfordulás)
kulcs:element/complexType/key/ selector és field xpath
idegen kulcs: element/complexType/keyref/ selector és field xpath

Saját típusdefiniálása:

```
<xs:simpleType name="eletkor_tipus">
<xs:restriction base="xs:integer">
<xs:maxInclusive value="120"/>
</xs:restriction>
</xs:simpleType>

<xs:element name="eletkor" type="eletkor_tipus"/>
```

37. Típus származtatás módjai és megadásuk xmlschema-ban

Származtatás módjai:
- meglévő elemi típusok megszorításokkal való kiegészítése
- elemi típusra épülő listaképzés (többértékű elem)
- elemi típusok uniója, ekkor a megadott típusok valamelyikéhez tartozhat az elem.

A szintaktika kétféle lehet:

```
<simpleType name="ujnev">     nevesített elemi típusra épül a lista
<list itemType="elemi_tipus" />
</simpleType>

<simpleType name="ujnev">     itt kerül pontosításra az alap, elemi típus is
<list>
        <simpleType> ...          leiras              ... </simpleType>
</list>
</simpleType>
```

38. Származtatott elemi típusok megadásai módjai

- meglévő elemi típusok megszorításokkal való kiegészítése
- elemi típusra épülő listaképzés (többértékű elem)
- elemi típusok unionja, ekkor a megadott típusok valamelyikéhez tartozhat az elem.

A szintaktika kétféle lehet:

```
<simpleType name="ujnev">     nevesített elemi típusra épül a lista
<list itemType="elemi_tipus" />
</simpleType>

<simpleType name="ujnev">     itt kerül pontosításra az alap, elemi típus is
<list>
        <simpleType> ...          leiras              ... </simpleType>
</list>
</simpleType>
```

39. Származtatott összetett típus megadási módjai

A származtatás a tartalom meghatározó elemen keresztül történik:

megszorítás:

```
<complexType name="ujnev">
<complexContent>
<restriction base=” …" >
<!-- megszorítások-->
</restriction>
</complexContent>
</complexType>
```

bővítés:

```
<complexType name="ujnev">
<complexContent>
<extension base=” …" >
<!–új elemek>
</extention>
</complexContent>
</complexType>
```

40. Absztrakt típusok és elemcsoportok megadása, használata xmlschema-ban

Az elemtípusok rugalmas kezelésére szolgálnak. Egy elem az 'abstract' jellemző 'True' értékre állításával tehető absztrakttá, globálisnak kell lennie. Az absztrakt elem nem fordulhat elő közvetlenül az XML dokumentumban, helyette valamely helyettesítése jelenik meg. Típusok is lehetnek absztraktak, ekkor a leszármaztatott típus jelenik meg helyette.

41. Névtér kezelés lehetőségei xmlschema-ban

Alap névtér kijelölése XMLSchema-ban: Az alapértelmezett névtér a séma gyökéreleménének 'targetNamespace' jellemzőjén keresztül állítható be.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://w3.org/2001/XMLSchema" targetNamespace="alap névtér url">
<!-- séma részletezése -->
</xs:schema>
```

42. Névtér kezelő parancsok xmlschema-ban

FORMDEFAULT tulajdonság szerepe a XMLSchema-ban: a névterek dokumentumbeli használatára vonatkozik a sémaleírás formdefault jellemzője, két fajtája van:

elementFormDefault: elemekre vonatkozik
attributeFormDefault: elemjellemzőkre vonatkozik.

Ezek két értéket vehetnek fel:
qualified: a célnévtérhez (targetNamespace) tartozó elemeket előtaggal kell ellátni
unqualified: a célnévtérhez (targetNamespace) tartozó elemeket nem kell expliciten előtaggal ellátni.

43. Sémák integrálási módjai névterek esetén

1.azonos munkanévtér esetén(INCLUDE)
<xs:include schemalocation="file:...">
2.azonos munkanévtér,verziókötés (redefine)
<xs:redefine schemaLocation="file:...">
3.Eltérő munkanévtér (IMPORT)
<xs:import schemaLocation="file:..." namespace="file:...">

44. Schematron modell célja, működési elve, megadása

Schematron működési elve: a Schematron dinamikus értékellenőrzési mechanizmus az XML dokumentumokra. Tetszőleges (XPath) alapú felületeket lehet vele vizsgálni. Nem teljesülő feltétel esetén hibajelzést ad vissza.

45. SAX api működési elve

Csak olvassa a dokumentumot, nem módosít; csak szekvenciálisan, egyszer olvassa át a dokumentumot; kis erőforrás igényű. A SAX feldolgozó és a gazdaprogram kommunikációja a CALLBACK mechanizmuson alapul. Ennek a lényege, hogy a gazdaprogramban szerepelni kell olyan feldolgozó moduloknak, amik alkalmasak az egyes felmerülő események kezelésére. Ezen felül a gazdaprogramnak közölnie kell a SAX modullal, hogy mely eseménynél mely modulja a felelős végrehajtó.

Működés: a kezelő modulok kötése után szekvenciális olvasás, ha struktúra esemény következik be, akkor a SAX modul megkeresi az illetékes gazdanyelvi komponenst, az lekezeli, majd vissza a SAX-hoz.

46. sax api: üzleti logika megadása????????

47. sax api: ContentHandler jellemzése

SAX API CONTENTHANDLER megadása:

public class osztályom implements ContentHandler{
    private Locator változó;
}
startDocument() - dokumentum feldolgozás kezdete
endDocument() - dokumentum feldolgozás vége
startPrefixMapping(String prefix, String uri) - névtér alias definiálása, prefix-alias uri-névtér
endPrefixMapping(String prefix) - névtér alias hatáskör vége
processInstruction(String target, String data) - feldolgozási direktíva, target: feldolgozó program azonosítása, data: utasítás
startElement(String namespaceuri, String localname, String rawname, Attributes atts) - elem kezdete, ahol namespaceuri:névtér, localname: elem neve, rawname: belső név, atts: befoglalt elemjellemzők listája
endElement(String namespaceuri, String localname, String rawname) - elem vége
characters(char[] ch, int start, int length) - szövegrész
ignorableWhitespace(char[] ch, int start, int length): kihagyható fehér karakterek
skippedEntity(String name) - kihagyott objektum

48. SAX API: feldolgozási események és kezelőjük, Locator

SAX API LOCATOR kezelés elemei:
public void setDocumentLocator(Locator loc){ definiálás
}
int getLineNumber() az eseményhez tartozó rész sorának sorszáma
int getColumnNumber() az eseményhez tartozó rész karakterpozíciója a soron belül

49. SAX API: lekérdezések megvalósítási logikája

Nagyjából ugyanaz, mint minden szekvenciális parser-nek. Element-enként olvassa be az XML-t, nem egyben az egészet. Leszármazol a defaulthandler osztályból, megvalósítod a startdocument, enddocument, startelement, endelement stb. függvényeket, és ezek szépen sorban hívódnak, ahogy az xml-en végigmegy a sax.

50. DOM API működési elve

DOM API működése elve: a kezelőfelület a memóriában felépíti a teljes dokumentumot, és a rendelkezésre álló metódusokon keresztül elvégezhető a tartalom átolvasása és módosítása. Szükség esetén fájlba írható a memóriában tárolt kép. A kimeneti állomány lehet azonos a bemenetivel, vagy lehet másik is. A DOM modellben az XML dokumentumot fa modellel írják le. A DOM API osztályok gyűjteménye, melyben a definiált adattagok az elemek jellemzőit írják le, a definiált metódusok a navigációra és a tartalom kezelésére szolgálnak. A DOM nagyobb végrehajtási költséggel jár, mint a SAX.

51. DOM API: üzleti logika megadása:???

52. DOM API: dokumentum objektum előállítása és kimentése

DOM dokumentum objektum előállítás lépései:

1. A DOM átolvassa a megadott bemeneti XML dokumentumot.
2. A memóriában felépíti a hozzá kapcsolódó objektum hierarchiát. A fa felépítése a SAX API-val történik. Kritikus pont a rendelkezésre álló memória végessége, ezért a hatékonyan feldolgozható XML dokumentum mérete is véges.

DOM objektum XML fileba mentésének módja:
Xerces serialization: formátum leíró létrehozása, célhely leíró létrehozása, serializer létrehozása, serialize metódus hívása.

53: DOM API: navigációs lépések, elemek közvetlen elérése

DOM navigációs elemek:
mozgás a gyerekek felé
navigáció a testvér csomópontok felé
átlépés a szülő csomópontra
DOM: elem lokalizálás módjai:
gyökérelem lekérdezése:

```
Document dok;
Node root = dok.getDocumentElement();
```

elem nevének és típusának lekérdezése:

```
Node elem;
if (elem.getNodeType()==Node.ELEMENT_NODE){
        if(elem.getNodeName().compareTo("ar")==0){}
        ...
}
```

54: DOM API: dom fa módosítási műveletek

DOM: elem érték módosítás módja: (setNodeValue)

```
Node elem;
ear = Integer.parseInt(elem.getTextContent());
String ujszov = String.valueOf(ear+nov);
elem.setNodeValue(ujszov);
```

55: DOM API: Node és alosztályainak jellemzése

Node osztály metódusai:

appendChild(Node)
cloneNode()
getAttributes()
getChildNodes()
getFirstChild()
getLocalName()
getNamespaceURI()
getNodeType()
getNodeValue()
getParentNode()
getPrefix()
getPreviousSibling()

getTextContent()
hasChildNodes()
hasAttributes()
removeChild()
replaceChild()
setTextContent()
setPrefix()

56: DOM API: attribútumok és szövegtartalom kezelése

Attribútum létrehozás: createAttribute(String)
lekérdezés: getAttributes()

Node típusú szövegtartalomnál: setNodeValue("string") metódus

57. xpath szerepe, xpath formátumai, kontextus

Az XPath az XML dokumentumot egy faként kezeli, mely nagyban hasonlít a DOM fa modelljére, de néhány ponton eltér tőle: nincs szimbólum, CDATA, DTD kezelés és minden csomóponthoz tartozik egy szöveges érték.

Az XPath kifejezések célja a fa csomópontjainak halmazából egy tetszőleges részhalmaz kiválasztása. Eredménye rendszerint egy részfa, mely lehet elemi vagy összetett szerkezetű. Konkrétan lehet: csomópont-halmaz, logikai értékű érték, numerikus érték, szöveges érték.

Kontextus – Az a környezet, amiben értelmezzük az XPath kifejezést.

58. xpath: tengelyek és megadásuk

Navigációs tengely – megadja a keresés fő irányát.

Az elérési lépés szintaktikája: tengely::szűrés[szelekció]

tengely:: csomópont_típus | elérési_útvonal [szűrési feltétel]
self::node() (.) maga a kontextus csomópont
child:: gyerek csomópontok, melybe nem tartoznak bele az elemjellemző és névtér csomópontok
descendant:: befoglalt csomópontok, gyerekek és azok gyerekei, az összes, tetszőleges mélységben befoglalt csomópontokat beleértve
descendant-or-self:: (//) a befoglalt csomópontok és maga a kontextus csomópont
parent::node() (..) a tartalmazó (szülő) csomópont
ancestor:: azon csomópontok, amik magukba foglalják a kontextus csomópontot
ancestor-or-self:: a befoglaló csomópontok és maga a kontextus csomópont
preceding:: azon csomópontok, melyek a dokumentum-sorrendben megelőzik a kontextus csomópontot és nem foglalják magukba a kontextus elemet, valamint típusuk nem elemjellemző és nem névtér
preceding-sibling:: azon csomópontok, melyek a dokumentum-sorrendben megelőzik a kontexus csomópontot, azonos a befoglaló elemük a kontexus befoglaló elemével, valamint típusuk nem elemjellemző és nem névtér
following:: azon csomópontok, melyek a dokumentum-sorrendben követik a kontextus csomópontot és nem foglalják magukba a kontextus elemet, valamint típusuk nem elemjellemező és nem névtér

following-sibling:: azon csomópontok, melyek a dokumentum-sorrendben követik a kontextus csomópontot, azonos a befoglaló elemük a kontextus befoglaló elemével, valamint típusuk nem elemjellemző és nem névtér
attribute:: (@) a kontextus csomóponthoz tartozó elemjellemzők halmaza
namespace:: a kontextushoz tartozó névtér leíró csomópontok halmaza.

59: xpath: elemhalmaz szurok és megadásuk: ???

60: xpath: operátorok, függvények

XPath relációs operátorok
+ - * div mod and or not() = != > <
| csomópont-halmaz egyesítés
XPath gyári függvények
- last(): kontextus mérete
- position(): aktuális csomópont sorszáma a szülőn belül (pozíciója)
- count(halmaz): csomóponthalmaz számossága
- name(): aktuális csomópont neve
- namespace-uri(): aktuális elem névtere
- concat(s1,s2..): string összefűzés
- string(kif): konverzió szövegre
- contains(s1,s2): rész-szöveg vizsgálat
- String-length(s1): szöveg hossza
- Sum(csomóponthalmaz): összeg
- Floor(n): egészrész
- Round(n): kerekítés egészre

61. XSLT működési modell, default minta

Bemenet: az input XML-hez készített dokumentumfa.
Kimenet: az eredményfa linearizálásával kapott XML dokumentum.
Feldolgozás:
1. XML állomány beolvasása.
2. A dokumentum-fa reprezentáció előállítása.
3. Elindul a fa gyökeréből lefelé.
4. Megnézi, van-e explicit minta ezen elemre:
   ha igen, végrehajtja a hozzátartozó parancsot
   ha nincs, végrehajtja az implicit parancsot.
5. Leáll.
XSLT default működési szabály:
1. Minden nem text elemnél továbblép a gyerek csomópontok felé (nem beleértve a névteret és elemjellemzőket)
2. Minden text elemnél kiírja mit tartalmaz és leáll az ággal.
3. Egyéb csomópontokat figyelmen kívül hagy.

62. XSLT: navigációs parancsok

A dokumentumfa egy megadott csomóponthalmazának kijelölése a
<xsl:template match=kif1 name=kif2 >
<!-- feldolgozó utasítások -->

</xsl:template>
paranccsal történik.
A fabejárás folytatásának parancsalakja:
<xsl:apply-templates select = "kif1" >
<!-- rendezés -->
</xsl:apply-templates>

63. XSLT: ciklus, elágazás parancsok

if / choose elemek:
If: feltételvizsgálat, akkor hajtja végre az igaz ág utasításait, ha a vizsgált kifejezés logikai értéke igaz - egyszeres elágazás.

```
<xsl:if test="kif">
        <!-- igaz ág -->
</xsl:if>
```

Choose: feltételvizsgálat, több különböző elemből választunk ki egyet - többszörös elágazás. Az első igaz ág fut le, egyébként otherwise.

```
<xsl:choose>
        <xsl:when test="kif1">
                <!-- tevékenységek -->
        </xsl:when>
        <xsl:when test="kif2">
                <!-- tevékenységek -->
        </xsl:when>
        <xsl:otherwise>
                <!-- tevékenységek -->
</xsl:choose>
```

XSLT: for-each-group elem: az XSLT csoportképzési kifejezése. Használatához paraméterként meg kell adni a feldolgozandó elemeket és a csoportképzési kifejezést. Lehetőség van csoport-szintű aggregációra, és elem szintű részletezésre.
<xsl:for-each-group select="elemek" group by="csoportképzési_kifejezés">
A feldolgozó magban a csoport elemeinek elérésére két szimbólum használható:
        current-group: az aktuális csoport elemeinek szekvenciája
        current-grouping-key: az aktuális csoporthoz tartozó csoportképzési kifejezés.
Az aggregációk végrehajtására az XSLT-XPath numerikus függvényei használhatók: avg() sum() max() min() count() .

64. XSLT: elem konstrukciós parancsok

```
<xsl:element name="nev">
        <!-- tartalom -->
</xsl:element>
```

Attribute: új elemjellemző létrehozása. Paraméter: select-a létrehozott jellemző értéke.
<xsl:attribute name="nev" select="kif" />
Text: egy egyedi, szöveg alapú csomópont létrehozása.

```
<xsl:text>
        <!-- tartalom -->
</xsl:text>
```

65. XSLT: változó, kiértékelés, operátorok, xPath szerepe

XSLT: változó használata: A változó (variable) olyan szimbólum, ami egy változó megértését, áttekinthetőségét segíti elő. Használatának indokai: beszédes jelölés az értékekre (PI <> 3.1416), formulák áttekinthetővé tétele a részkifejezések kiemelésével, leírás egyszerűsítése az ismétlődő rész helyettesítésével. Létezik globális (teljes XSLT-ben) és lokális (csak a definíciós blokkjában) változó. Akkor lesz globális, ha a gyökér alatt van definiálva.

<xsl:variable name="nev" as="adat_tipus"> ... </xsl:variable>

XPath célja, kifejezés általános alakja: az XPath kifejezések célja a fa csomópontjainak halmazából egy tetszőleges részhalmaz kiválasztása. Eredménye rendszerint egy részfa, mely lehet elemi vagy összetett szerkezetű. Konkrétan lehet: csomópont-halmaz, logikai értékű érték, numerikus érték, szöveges érték. A kiválasztás mintaillesztés alapján történik.

66. XSLT: saját függvény és nevesített template

XSLT: saját függvény kezelése, használata: a függvény hasonlít a névvel ellátott sémákra, viszont meghívása minden olyan helyről megtörténhet ahol ezt az adattípus illesztés megengedi. A függvény nevének névtérrel kiterjesztett alakúnak kell lennie. Visszatérési értéke a végrehajtása során előállított, nincs külön ezt kijelölő utasítás.

<xsl:function name="nevter:nev" ... > ... </xsl:function>

A függvény meghívása a nevével történik, a hívás XPath kifejezésben is szerepelhet. Függvényekhez is köthető paraméterátadás, a param elemmel, a hívási függvénynevet követő zárójelben kell lenniük az aktuális paramétereknek.

XSLT: nevesített minta kezelése (paraméterezett formulák), használata: a transzformációs sémát nem mintához kötjük, hanem azonosító nevével hívjuk meg.

<xsl:template name="azonosito" ...>...</xsl:template>

Meghívása: <xsl:call-template name="azonosito">...</xsl:call-template>

Paraméterek kijelölése: <xsl:param name="parameter">...</xsl:param>

A param elem jellemzői: as (adattípusa), select (default paraméter), required (kötelező megadni).

A híváskor az aktuális paramétereket a call-template elem gyerekeként szerepeltetik, a paraméterátadás utasítása a with-param elem: <xsl:with-param name="parameter" select="ertek"> .. </xsl:with-param>

67. xQuery működési modell

3 fázis: előfeldolgozás, statikus elemzés, dinamikus elemzés.

Előfeldolgozás: a bemeneti XML dokumentumokból létrejön a dokumentumok fa modellje. Ennek lépései:

Az XML linearizált alakjának elemzése - parsing

Infoset modell készítése
Infoset modell validálása
A validált Infoset elemekből érvényes információs elemkészlet építése
A dokumentum fa modelljének előállítása.

Statikus elemzés: szintaktikai elemzés, ami során egy normalizált műveleti gráf jön létre.
Az XQuery parancs szintaktikai elemzése, kisebb részekre bontás
Az előállított műveleti halmazból műveleti gráf építése
A műveleti gráf kiértékeléséhez szükséges információk összegyűjtése
Kontextus-leíró struktúra a begyűjtött paraméterekből
Normalizált műveleti gráf előállítása.

Dinamikus kiértékelés: a műveleti gráf, a létrejött XML modell és a kontextus leírók alapján kimenet elkészítése.
A műveleti gráf alacsonyabb szintű algoritmus fává alakítása
Kifejezések kiértékelése
Előállnak a válasz dokumentum alkotó elemei, amiket kimeneti atomoknak hívnak
Az Infoset készletből XDM fa építése
Az XDM fa sorosítása, linearizálása, ennek végén áll elő a kimeneti dokumentum.

68. xQuery: alap parancsok

FOR elem	ciklus
LET elem	értékadás
ORDER BY elem	rendezés
WHERE elem	szelekció
RETURN elem	projekció

69. xQuery: ciklus, elágazás parancsok

Ciklus szervezése: lényege, hogy a változó végigfut a megadott halmaz elemein.

```
FOR $valtozo IN lista
RETURN
    kifejezes
```

A ciklusokat lehetséges egymásba ágyazni, ekkor a belső iteráció a külső iteráció minden értékére lefut.

if elemek: ha a kimeneti állomány egyes részei feltételtől függően változnak, akkor feltételes kiértékeléssel építjük fel a kimeneti dokumentumot.

```
return
    {
    if (kifejezes1)
        then kifejezes2
        else kifejezes3
    }
```

70. xQuery: elem konstrukciós parancsok

xQuery elem felvitel: 3 fontosabb szintakszis változat:

Gyerekcsomópont felvitele:

```
insert nodes forras_csomopont as (first|last) into cel_csomopont
```

Elem elé történő beszúrás:

```
insert node forras_csomopont before cel_csomopont
```

Elem mögé történő beszúrás:

insert node forras_csomopont after cel_csomopont

71. xQuery: változó, kiértékelés, operátorok

xQuery: változó használata: XQuery-ben a változók előtt $ jel szerepel. A változók szerepe, hogy a feldolgozandó csomóponthoz, vagy csomóponthalmazhoz rendelhetünk vele egy szimbólumot.
$változó
A létezik és minden logikai kvantorok megadása:
every $változó in halmaz satisfies kifejezés minden
some $változó in halmaz satisfies kifejezés létezik.

Operátorok:

```
eq =
ne !=
lt <
le <=
gt >
ge >=
```

72. xQuery: saját függvény használata, csoportképzés

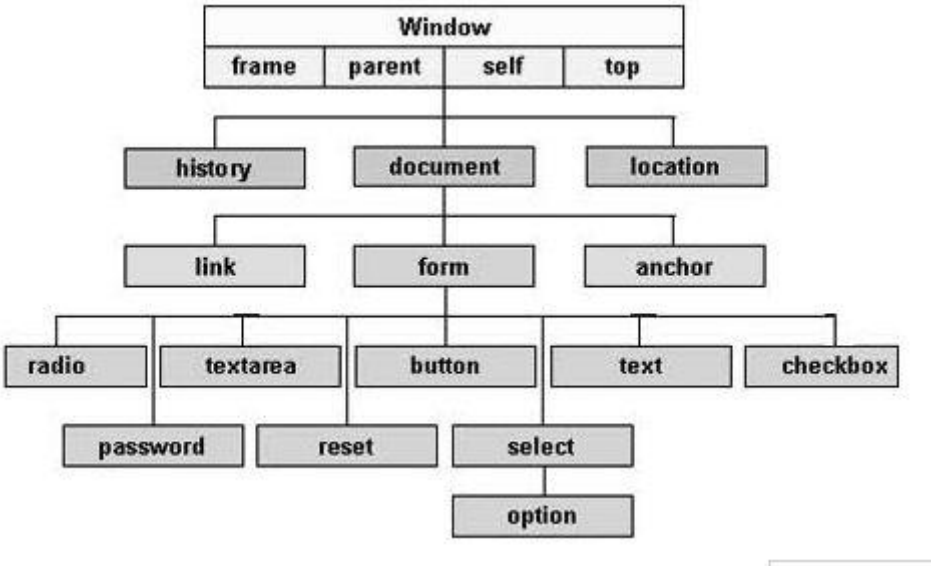
xQuery: saját függvény kezelése, használata:
A változók csak egyszer vehetnek fel értéket, és a ciklusokat rekurzív hívásokkal lehet megoldani. A saját függvény létrehozásához szükség van egy saját névtérre is.
declare namespace prefix=kifejezes;

declare function prefix:fuggvenynev ($p1 as t1, ...) as rtip
p1 változó, rtip a visszatérési érték típusa
{
    utasitasok ...
    return kifejezes; a visszatérési érték
};

xQuery: csoportképzés megvalósítása: csak kerülő úton megoldható, nincs group by. A csoportosításhoz a distinct-values() függvény használható, ami az argumentumában megadott értékhalmazból olyan értékhalmazt állít elő, amiben nem fordul elő ismétlődés. Az aggregált értékek lekérdezéséhez használatos függvények: min() max() sum() count() avg() . Az aggregációs függvényeknél a függvény paraméterében explicite ki kell jelölni a feldolgozandó halmazt.

73. XML Encryption működési modellje, elemei

74. JavaScript DOM modellje, elemei



75. AJAX és XML

76. WebService és XML

Web Service egy olyan hálózaton elérhető informatikai rendszer, melynek interface-e WSDL szabvány szerinti leíróval definiált és mely a SOAP szabványnak megfelelő üzenetekkel kommunikál.

A WSDL leíró egy XML formátumú dokumentum, melynek felépítését szabványosított XSD rögzíti.

77. xForms jellemzése

Modulárisan tervezhetünk benne, MVC (Model-View-Controller) alapon. Az adatmodell független a megjelenítéstől. Az üzleti logikai szeparált, átláthatóbb. Kevesebb kódolást igényel, nem kell írni javascript kódot <script/> . A meglévő korszerű technológiákra épül, pl. felhasználja az AJAX technológiát. Utasításainak nyelve XML. A kimeneti dokumentum HTML+JavaScript formátumú.

A jegyzetet készítette:

Huzynets Erik gazdaságinformatikus hallgató (Miskolci Egyetem)

Sass Péter jegyzeteit felhasználva

Miskolc, 2013.február 1.