МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДНІПРОВСЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

О.О. Шумейко

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни **«Технології створення WEB - застосувань»**

для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньо-професійною програмою
«Інженерія програмного забезпечення»
зі спеціальності 121 – «Інженерія програмного забезпечення»
галузі знань 12 – «Інформаційні технології»

ЗАТВЕРЖЕНО:

редакційно–видавничою секцією

науково–методичної ради ДДТУ

_____2019 р., протокол №

Кам’янське

2019

Конспект лекцій з дисципліни «Технології створення Web-застосувань» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення» галузі знань 12 – «Інформаційні технології» / Укладач О.О.Шумейко.– Кам'янське: ДДТУ, 2019– 124 с.

Укладач:
доктор технічних наук, професор Шумейко О.О.

Рецензент:
завідувач кафедри прикладної математики
доктор технічних наук, професор Самохвалов С.Є.

Відповідальний за випуск: зав. каф. ПЗС Шумейко О.О.

Затверджено на засіданні кафедри програмного забезпечення систем (протокол № від «__» ___ 2019 р.)

Коротка анотація видання. Конспект лекцій складено відповідно до освітньо-професійної для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення» галузі знань 12 – «Інформаційні технології» та робочої програми з дисципліни «Технології створення Web-застосувань» і відповідає вимогам модульного засвоєння курсу. Викладено основні теми дисципліни, що присвячені основам програмування на алгоритмічній мові PHP та XML-технології. Містить контрольні запитання до усіх розділів лекційного матеріалу дисципліни; окремі теми супроводжуються прикладами розв'язання типових задач.

ЗМІСТ

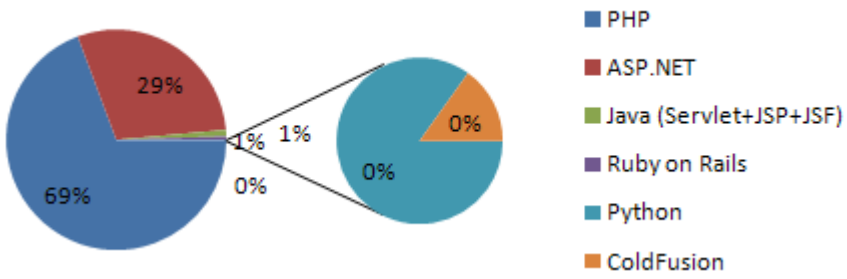
Вступ

Ті три кити, на які спирається програмування інтернет додатків є мова гіпертекстової розмітки HTML, мова динамічного управління контентом JavaScript та мова серверної взаємодії та зв'язку з СУБД. Ці три кити породили цілу низку програмних продуктів, які активно використовуються у сучасному інтернет програмуванні. Вивчення дисципліни «Технології створення Web-застосувань» спирається на те, що студенти володіють html та css, мають навички роботи з СУБД, знайомі з SQL.

Для ефективної роботи з Web-сервером вибрано мову PHP (Hypertext Preprocessor - Препроцесор Гіпертексту). PHP - мова програмування, яка спеціально розроблена для написання веб-додатків, (скриптів, сценаріїв), що виконуються на Web-сервері. Синтаксис мови багато в чому ґрунтується на синтаксисі C та Perl. PHP-скрипти виконуються на стороні сервера і користувачу не відомо, чи отримує він звичайний HTML-файл, чи результат виконання скрипта. PHP здатний генерувати і перетворювати не тільки HTML-документи, але й зображення різних форматів - JPEG, GIF, PNG, PDF файли і Flash, формувати дані в будь-якому текстовому форматі, включаючи XHTML і XML.

Вибір PHP зумовлений його популярністю, що і наведено у даній таблиці згідно статистики http://www.acunetix.com/blog/articles/statistics-from-the-top-1000000-websites/

Name	PHP	ASP.NET	Java	Ruby on Rails	Python	ColdFusion
Percentage	69.27%	29.24%	0.96%	0.48%	0.022%	0.003%



Мова програмування PHP, особливо в зв'язці з популярною базою даних MySQL є оптимальним варіантом для створення інтернет-сайтів різної складності.

MySQL	MariaDB	PostgreSQL	MongoDB	CouchDB
65%	12%	11%	11%	1%

(http://habrahabr.ru/company/jelastic/blog/186256/)

Частина 1. PHP

Тема 1: Основи програмування на PHP.

PHP-інструкції пишуться прямо в HTML-документі , але як відокремити PHP від HTML? За допомогою спеціальної угоди вказуємо, де починається і закінчується код. Все інше сприймається як HTML-документ. Найчастіше для позначення початку і кінця PHP-інструкцій використовується формат:

```
<?php
        КОД PHP
?>
```

Все, що розташовується між тегами <?php ?> сприймається сервером як PHP-код і обробляється відповідним чином. Все інше сприймається як HTML-документ і передається клієнтові без змін і обробляється вже в браузері. Існують і інші можливості об'явлення php коду, про що піде мова на лекції.
Мова програмування PHP стоїть на плечах мови С, тому знайомство з мовою С дозволяє досить легко почати програмування на PHP. Але тут існують особливості, які треба знати.

Підключення файлів

Багаторазове використання коду в PHP реалізовано через підключення файлів. Підключення файлів можна сприймати і як ще один спосіб впровадження PHP-коду у веб-сторінку, що може бути реалізовано наступним чином:

```
include('/filepath/ filename');

include_once('/filepath/filename');

require('/filepath/filename');

require_once('/filepath/filename');
```

перші дві функції при виявленні помилки (наприклад, відсутній файл, що підключається) видають попередження і продовжують виконання сценарію. Функції require()/ reguire_once() видають повідомлення про критичну помилку, і подальша робота переривається.
Різниця між include()/require() і include_once()/require_once() полягає в тому, що друга пара функцій гарантує, що вказаний файл буде підключений до поточного файлу тільки один раз. Іноді це буває необхідно, щоб двічі не підключати файл з критично важливим кодом, який повинен бути тільки в єдиному екземплярі.

Друк

Для виведення інформації на екран / вікно браузера є дві функції: echo та print. Існує два варіанти виклику функції echo:

```
echo ("Привіт, ", "Світ!");
echo  "Привіт, ", "Світ!";
```

У першому випадку рядок, що виводиться, вказується в дужках, а в другому випадку рядок вказується через проміжок. Обидва виклику еквівалентні, і можна використовувати той, який більше подобається. Але можна і не обмежуватися одним рядком, і виводити відразу по кілька рядків, розділяючи їх комами.

Функція print() відрізняється тим, що може повертати значення, яке вказує на успішність виконання завдання. Якщо функція повернула 1, значить, друк пройшла вдало, інакше функція поверне 0. Але є обмеження - друкувати можна тільки один рядок, тобто не можна написати два рядки через кому.
Для форматованого виводу існує функцію printf(), яка прийшла у PHP з мови програмування С і має такі ж самі управляючі параметри.

Змінні
Змінна - це область пам'яті, в якій можна зберігати якісь значення і звертатися до цієї пам'яті по імені. Змінні в PHP мають наступні властивості:

- всі імена змінних повинні починатися з символу долара ($), і по такому знаку інтерпретатор визначає, що це змінна;
- значення змінної - це значення, яке було записано в неї останнім;
- змінна починає існувати з моменту присвоєння їй значення або з моменту першого використання. Якщо використання починається раніше присвоєння, то змінна буде містити значення за замовчуванням;
- змінної не призначається певний тип. Тип визначається зберігаються значенням і поточною операцією.

Першим символом після знаку $обов'язково повинна бути літера.
Для запису значення в змінну використовується знак рівності.

```
$Ім'я змінної  = значення;
```

У PHP є три основних типи змінних - числовий, строковий і логічний. З першими двома все зрозуміло. Якщо це число, то змінна містить числове значення, при оголошенні рядків значення обмежується подвійними або одинарними лапками:

```
$str = 'Рядок';
$str = "Рядок";
```

Між цими двома записами є суттєва різниця. Нехай є

```
$index = 10;

$str1 = 'Index = $index';

$str2 = "Index = $index".
```

Якщо вивести на екран значення змінної $str1, то на екрані побачимо текст Index = $index, а у другому випадку Index = 10. Якщо використовуємо подвійні лапки, то всі змінні всередині рядка будуть інтерпретуватися за значенням, тобто, змінна буде замінюватися її значенням. Якщо використовуються одинарні лапки, то текст виводиться повністю, без перевірки і зміни.

Чутливість
Мова PHP не чутливий до проміжків, переводів рядка або знакам табуляції. Це означає, що ви можете розділяти одну команду на кілька рядків або відокремлювати змінні, значення або оператори різною кількістю проміжків:

```
<? PHP
    $Індекс = 1+2;
    $Індекс = 1 + 2;
    $Індекс =
        1
```

```
    +
        2;
?>
```

З точки зору PHP весь цей код коректний і буде правильно відпрацьований
Кожна команда в PHP закінчується крапкою з комою (;).
Мова PHP чутлива до регістру символів. Це означає, що імена змінних, написані в нижньому регістрі і у верхньому, будуть сприйматися як різні.

```
<?php

    $index = 10;

    $Index = 20;

    print($index};

    print($Index};

?>
```

В результаті ми побачимо спочатку число 10, а потім 20, тому що змінні $index та $Index будуть сприйматися як різні через відмінності в регістрі першої літери.

Але це стосується тільки змінних. Оператори мови PHP можуть бути написані в будь-якому регістрі. Наприклад

```
<?php $index=l;

If ($index==l)

    PRINT('true');

ElsE

    Print('false');

?>
```

Тут ключові слова, містять літери в різних регістрах, і це не є помилкою.

Область видимості
Всі змінні і функції мають область видимості, тобто існують ділянки коду, де їх значення доступно. За межами області видимості змінні автоматично знищуються.
Всі змінні в мові PHP є глобальними, якщо вони не оголошені всередині якої-небудь функції. Це означає, що якщо змінну оголосити на самому початку файлу, то вона буде існувати протягом обробки всього файлу і її можна використовувати в будь-якому місці. В даному випадку область видимості - поточний файл, і змінна знищиться при виході з нього.

Управління виконанням програми

Як вже було зазначено, багато операторів PHP прийшли з C, так, наприклад оператор умовного переходу

```
if (умова)
  Дія 1;
else
  Дія 2;
```

Якщо умова, зазначена в дужках, вірна, то виконається дія 1, інакше буде виконано дію 2, наприклад:

```
$index = 0;
if ($index> 0) print ("Index> 0");
else
  print ("Index = 0");
```

В даному випадку відбувається така перевірка: якщо змінна $index більше нуля, то виводиться перше повідомлення, інакше виводиться друге повідомлення, яке стоїть після ключового слова else.

Оператор вибору switch:

```
switch (Змінна) {
        case Значення 1;
                Оператор 1;
                break;
        case Значення 2;
                Оператор 2;
                break;
        [Default: Оператор]
}
```

У даному випадку змінна порівнюється зі значеннями, які вказані після ключового слова case. Якщо значення змінної дорівнює Значення I, то будуть виконані всі оператори цього блоку до ключового слова break. Зазначимо, що якщо після порівняння if виконується тільки один оператор (для виконання декількох операторів їх треба об'єднати фігурними дужками), то в даному випадку може виконуватися будь-яку кількість операторів.

Якщо під час виконання перевірок жодне значення не підійшло, то буде виконаний оператор, який стоїть після ключового слова default. Це слово є необов'язковим, але може бути присутнім для виконання будь-яких дій. Наприклад, якщо значення змінної не підійшло ні під одну з перевірок, то можна вивести повідомлення про помилку.

Цикли таку ж самі як і у C.

Цикл за перечисленням

```
for (начальне значення лічильника; останнє значення; крок зміни
лічильника);
```

цикл з передумовою

```
while (умова) дія;
```

та цикл з післяумовою

```
do дія while (умова);
```

З PERL до PHPприйшов оператор перебору foreach, про який мова піде трохи пізніше.

Альтернативна запис управляючих структур

Управляючі структури можуть бути записані в альтернативному виді. Це дозволяє не тільки виводити змінні в шаблоні, а й прописувати умови виведення певних даних безпосередньо у тілі html-сторінки.

Цикл foreach можна записати так:

```
<? foreach ($array as $item): ?>

    <li><?=$item?></li>

<? endforeach; ?>
```

В даному випадку замість фігурних дужок використовується закриваючий вираз endforeach. Зауважимо, що відкриваюча конструкція завершується двокрапкою, а закриваюча - крапкою з комою.

Також в альтернативному варіанті можна записати структури: for, while, if-else, switch-case.

Цикл for

```
<? for ($item = 1; $item <= 10; $item ++): ?>

    <li><?= $item ?></li>

<? endfor; ?>
```

Цикл while

```
<? while ($item < 10): ?>

    <li><?= $item ?></li>

<? endwhile; ?>
```

Оператор умовного переходу if-else

```
<? if ($item > 10): ?>

    Значення більше десяти.

<? elseif ($item < 10): ?>

    Значення меньше десяти.

<? else: ?>

    Значення дорівнює десяти.

<? endif; ?>
```

У разі використання оператора switch потрібно звернути увагу на те, що не можна розривати умову і перший вираз.

```
<? switch ($item):
        case 1: ?>
                Значення дорівнює 1.
                <? break; ?>
        <? case 2: ?>
                Значення дорівнює 2.
                <? break; ?>
        <? case 3: ?>
                Значення дорівнює 3.
                <? break; ?>
        <? default: ?>
                Значення не відомо.
<? endswitch; ?>
```

Масиви

Масив - це список значень, доступ до яких можна отримати за допомогою однієї змінної. Доступ до елементів масиву забезпечується за допомогою індексу, в якості якого може виступати слово або число. Якщо це числовий індекс, то він нумерується з нуля.

Масиви іменуються так само, як і змінні, але після імені вказуються квадратні дужки. У наступному прикладі в масив додаються три слова "торт", "хліб" і "морква ":

```
$food[] = "торт";
$food [] = "хліб";
$food[] = "морква";
```

Щоб вивести на екран нульовий елемент, напишемо ім'я змінної, а в квадратних дужках відповідний індекс:

```
print ("<P> $food[0]");
```

У даному прикладі під час додавання елементів до масиву ми нічого не вказували в квадратних дужках, тому кожному новому елементу присвоюється черговий індекс, який збільшується на одиницю. Під час створення масиву ви можете явно вказати індекс кожного елемента. Наприклад:

```
$food[0] = "торт"; $food [1] = "хліб"; $food [2] = "морква";
```

Цей спосіб зручніше, тому що контролюються індекси, які використовуємо. При цьому можна оголошувати елементи в будь-якому порядку і необов'язково використовувати послідовно. Наприклад:

$food [3] = "торт"; $food [9] = "хліб"; $food [2] = "морква";
Подивимося, що вийде, якщо додати новий елемент до цього масиву
без вказівки індексу:

$food[] = "картопля";

Цьому елементу буде привласнений індекс, який виявиться на 1 більше максимального з існуючих, тобто число 10.
У наступному прикладі показано, як можна створювати масиви, в яких в якості індексів виступають символи:
$food["ca"] = "торт"; $food ["b"] = "хліб";
$food ["cr"] = "морква"; echo ($food ["b"]);
Як бачите, різниця між числовими і символьними індексами невелика.
Більш інтелектуальний спосіб створення масивів - використання функції array ():
$food = array ("торт", "хліб", "морква");
У даному випадку змінна масиву виглядає так само, як і будь-яка інша змінна. Їй присвоюється масив, створений функцією array (), у якого в дужках перераховані елементи масиву. Цим елементам будуть присвоєні індекси, починаючи з 0 і до 2.
За допомогою функції array () можна створювати масиви і з символьними індексами. Наприклад:
$food = array ("ca" => "торт", "b" => "хліб", "cr" = "морква")
Відповідність індексу імені елемента масиву потрібно писати наступним чином:
"Індекс" => "значення"
Так як індекси в масиві можуть йти не по порядку, то нам необхідний спосіб створення циклу, в якому можна переглянути всі елементи. Для цього найпростіше використовувати цикл foreach, який ми ще не обговорювали:

foreach (array as [$key =>] $value)
оператор;

Якщо вказана змінна $key (ім'я змінної може бути іншим), то в тілі циклу через цю зміну можна отримати доступ до індексу елемента. Через змінну $value можна отримати доступ до значення елемента масиву.
Розглянемо приклад, в якому виводяться на екран значення індексів і значення всіх елементів масиву:

```
foreach $food as $idx => $val) {
        print ("<P> index: $idx <BR> value: $val");
        }
```

Робота з текстовими рядками
Робота з текстом є вузловою у веб-додатках. Розглянемо декілька важливих функцій.
Функція substr
Дуже часто нам потрібно отримати певну ділянку рядка. Для цього використовується функція substr ():
string substr (string string, int start [, int length])
У цієї функції три параметри:

- рядок, підрядок який необхідно отримати;
- номер символу, починаючи з якого потрібно повернути символи;
- необов'язковий параметр, який вказує на кількість повертаються символів, починаючи з позиції в другому параметрі. Якщо цей параметр порожній, то будуть повернуті всі символи до кінця рядка, починаючи з позиції start.

При вказівці позиції символів треба враховувати, що символи у рядку нумеруються з нуля. Як результат функція substrt) поверне вказаний набір символів.

Функція strlen

Функція повертає довжину рядка. Як параметр вказуємо рядок або змінну, що містить рядок, а на виході отримуємо довжину.

Функція strpos

Нерідко виникає необхідність розібрати рядок на складові або знайти в рядку певний символ чи сполучення символів. Для цього використовується функція strpos(), в яку передаються три параметри:

- Рядок, в якому потрібно шукати підрядок;
- Підрядок, який треба знайти;
- Позиція, починаючи з якої потрібно шукати (необов'язковий параметр). Якщо параметр не зазначений, то пошук підрядка здійснюватиметься, починаючи з першого символу рядка.

Контрольні питання.

1. Чим відрізняються оператори виводу інформації echo() та print()?
2. Яка різниця між виводом інформації у одинарних і у подвійних лапках?
3. В якому сенсі розуміється глобальність зони видимості змінних?
4. Чи може лише частина масиву індексована символами?
5. Яка різниця між змінними $str=”Hello” та $str=’Hello’.
6. В чому сенс альтернативного запису управляючих конструкцій PHP.

Тема 2:Використання PHP для роботи з HTTP.

Взаємодія клієнта і сервера забезпечується обміном інформацією між ними. Для передачі інформації від клієнта серверу використовуються елементи форми html. Елементи форми, такі як текстове поле, прапорець, випадаючий список та ін. використовуються для отримання інформації від клієнта, а кнопка submit дозволяє отриману інформацію переслати серверу для подальшої обробки.
Для отримання даної інформації на сервері PHP читає масиви суперглобальних змінних $_GET і $_POST.
Після отправки форми з полями

```
<input type="text" name="txt">
<input type="password" name="passw">
<input type="hidden" name="hid" value="">
```

в залежності від методу передачі даних на сервері будуть створені наступні змінні околу:

метод GET:
$_GET["txt"], $_GET["passw"] ,$_GET["hid"]
метод POST:
$_POST["txt"] ,$_POST["passw"], $_POST["hid"].

Назва змінної співпадає зі значенням параметру name тегу <input>.
Нехай в ці поля необхідно ввести первісні значення у сценарії. В цьому випадку можливі наступні проблеми.
Якщо у рядку є проміжки, то використання дужок обов'язково. Нехай

```
$str = 'Всім вітання';
echo '<input type="text" name="txt" value=' . $str . '>';
```

тоді в результаті поле txt буде мати текст 'Всім', а не 'Всім вітання'. Правильно буде так:

```
$str = 'Всім вітання';
echo '<input type="text" name="txt" value="' . $str . '">';
```

Якщо у рядку є лапки, то їх треба замінити їх сутностями, HTML-еквівалентами.

```
$str = 'Група "Океан Ельзи"';
echo '<input type="text" name="txt" value="' . $str . '">';
```

то в результаті поле txt буде мати текст 'Група', а не 'Група "Океан Ельзи"'. Правильно буде так:

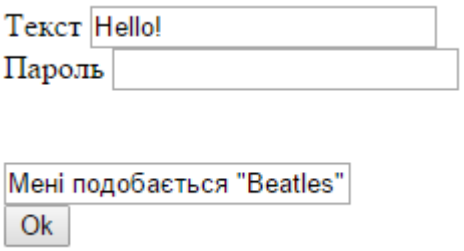
```
$str = 'Група "Океан Ельзи"';
$str = htmlspecialchars($str);
echo '<input type="text" name="txt" value="' . $str . '">';
```

Приклад.
Файл text.html

```
<html>
<head>
    <title>Метод GET</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
```

```
</head>
<body>
	<form name="TextForm" action="text.php" method="GET"/>
	Текст <input type="text" name="Text"/><br/>
	Пароль <input type="password" name="Pass"/><br/>
	<input type="hidden" name="Hid" value="Top secret"/><br/>
	<?php
	$str='Мені подобається "Beatles"';
	$str= htmlspecialchars($str);
	echo '<input type="text" name="TextScript" value="'.$str. '">';
	?>
	<br/>
	<input type="submit" value="Ok"/>
	</form>
</body>
</html>
```

Виглядає наступним чином



Файл text.php

```
<html>
<head>
	<title>Метод GET</title>
	<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
<?php
	$text=$_GET["Text"];
	$pw=$_GET["Pass"];
	$h=$_GET["Hid"];
	echo "Текст <b>",$text,"</b><br/>";
	echo "Пароль <b>",$pw,"</b><br/>";
	echo "Скрите поле <b>",$h,"</b><p/>";
	if(isset($_GET["TextScript"]))
	{
	$ts=$_GET["TextScript"];
	echo "Текст з скрипта <b>",$ts,"</b><br/>";
	}
?>
</body>
</html>
```

Текст **Hello!**
Пароль
Скрите поле **Top secret**

Текст із скрипта **Мені подобається "Beatles"**

Розглянемо використання radiobutton- елементів одинарного вибору

```
<html>
<head>
<title>Radio</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<form name="RadioForm"  method="GET">
<input type="radio" name="voice" value="one"/>Це раз<br/>
<input type="radio" name="voice" value="two"/>Це два<br/>
<input type="radio" name="voice" value="three"/>Це три<br/>
<input type="submit" value="Ok"/>
</form>
</body>
</html>-
<?php
  switch($_GET['voice'])
        {
        case 'one': echo 'Зараховано раз'; break;
        case 'two': echo 'Зараховано два'; break;
        case 'three': echo 'Зараховано три'; break;
        default: echo 'Щось треба вибрати';
        }
?>
```

Це раз
Це два
Це три
Ok

- Щось треба вибрати

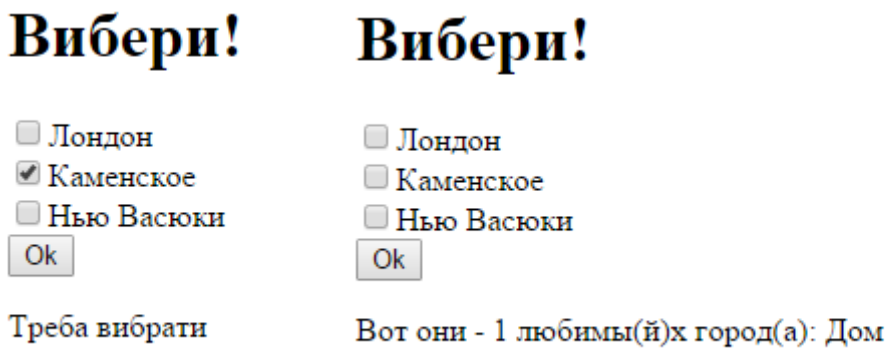
Це раз
Це два
Це три
Ok

- Зараховано два

Розглянемо використання прапорців

```
<html>
<head>
<title>Check</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
<h1>Выбирай!</h1>
<form  method="POST">
<input type="checkbox" name="CheckForm[]" value=" Увидеть ... и умереть "/>Лондон<br/>
<input type="checkbox" name="CheckForm[]" value=" Дом "/>Каменское<br/>
<input type="checkbox" name="CheckForm[]" value=" Ха-ха "/>Нью Васюки<br/>
<input type="submit" value="Ok"/>
</form>
```

```
</body>
</html>
<?php
$city=$_POST['CheckForm'];
if (empty($city))
	{
		echo "Нужно выбрать";
	}
		else
	{
	$num=count($city);
	echo "Вот они - $num любимы(й)х город(а): ";
	for($i=0;$i<$num;$i++)
	{
		echo $city[$i];
	}
	}
function IsCheck($chk,$val)
{
	if(!empty($_POST[$chk]))
	{
	foreach($_POST[$chk] as $chkval)
		{
		if($chkval==$val)
			{
			return true;
			}
		}
	}
	return false;
}
if(IsCheck('CheckForm',' Дом '))
echo "<p/>Патриот!";
?>
```



Cookie

В силу того, що протокол HTTP є протокол без пам'яті, то для того щоб зберегти проміжні дані в давні часи становлення Інтернет, фірма Netscape Communications створила таку річ, як cookie. Cookie не повинно перевищувати 4 kb. Раніше були обмеження на кількість відісланих cookie з однієї адреси, зараз ці обмеження зняті.

При зверненні до сервера, клієнту надходить вказівка на установку cookie, тобто на значення cookie сервер буде реагувати тільки після перезавантаження сторінки, тобто, коли cookie буде встановлена і клієнт ЗНОВУ звернеться до сервера.
У PHP для роботи з cookie існує спеціальна функція

int setcookie (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]])

1. Ім'я cookie. Тільки латинські літери, цифри, символ підкреслення і дефіс. Всі інші символи перетворюються на символ підкреслення.
2. Значення параметра
3. Дата закінчення терміну придатності
4. Шлях, який визначає, в якій частині домену може використовуватися даний cookie
5. Домен
6. Вказівка, що дані cookie повинні передаватися через безпечне з'єднання HTTPS

Cookie: створення

```
set cookie ("TestCookie", $value);
set cookie ("TestCookie", $value, time () + 3600);
// Період дії cookie 1:00
set cookie ("TestCookie", $value, time () + 3600, "/ doc /", ". site.com", 1);
```

Першим ЗАВЖДИ повинно приходити cookie, якщо, наприклад, маємо запис

```
echo "Hello";
$Color = "gold";
setcookie ("TestCookie", $color);
```

то це буде помилка, оскільки до cookie вже відіслано повідомлення (echo), тобто заголовок вже сформований.
Зауважимо, що якщо файл написаний у кодуванні UTF-8 з BOM (Byte order mark), то теж буде помилка.

```
<?php setcookie ("TestCookie", 'test')?>
```

У кодуванні 866 сторінки отримаємо

```
я╗┐ <?php setcookie ("TestCookie", 'test')?>
```

Зауважимо, що блокнот Windows завжди зберігає utf-8 с BOM. Тому використовувати блокнот для редагування php файлів не бажано.

Cookie: читання

```
echo $_COOKIE ["TestCookie"];
// Створення масиву з різних cookie
while (list ($name, $value) = each ($_ COOKIE)) {
// Для послідовності значень
$Array [] = $value;
// Для асоціативного масиву
$Array [$name] = $value;
}
```

Cookie: масиви

```
// Створюємо масив
$Array = array (
"Name" => "Alex",
"Login" => "Admin",
"Password" => 1
);
// Упаковуємо в рядок
$Str = serialize ($array);
```

// Зберігаємо в cookie
set cookie ("TestCookie", $str, time () + 3600);
// Зчитуємо в масив
$Array = unserialize ($_ COOKIE ['user']);

Cookie: видалення

Нехай

```
<?php
        setcookie ("name", "Alex");
        if (isset ($_COOKIE ["name"])) echo $_COOKIE ["name"];
?>
```

Після запуску цього скрипта у вікні браузера нічого не з'явиться - сервер тільки послав заявку на установку кукі. Перевантажуючи ми побачимо повідомлення. Закриємо, відкриємо сторінку, все буде відображатися, поки не закриємо браузер.
setcookie ("name", "Alex", time () + 60);
// Встановлюємо час життя cookie 60 сек.
Тепер протягом зазначеного часу кукі будуть жити незалежно від стану браузера.

Контрольні питання.

1. Що є BOM при кодуванні UTF-8?
2. Яка різниця між cookies та сесією?
3. В чому різниця між передачею даних методом GET і методом POST?
4. В чому різниця між $HTTP_GET_VARS та $_GET?
5. Яка різниця між передачею даних з множини radio-button та множини прапорців?
6. Який максимальний розмір cookie?

Тема 3: Застосування регулярних виразів для перевірки та валідації даних.

Регулярні вирази – це система синтаксичного розбору текстових фрагментів по формалізованому шаблону, яка заснована на системі запису зразків для пошуку. Зразок (англ. pattern), що задає правило пошуку або, іншими словами, «шаблон» чи «маску».

Загальна задача механізму регулярних виразів - знаходити або не шукати збіги у рядку або частини рядка з шаблоном. У PHP існує кілька функцій для роботи з регулярними виразами: ereg(), ereg_replace(), eregi(), ereg_replacei() і split().

У PHP, для виконання перевірки на відповідність регулярному виразу, часто використовують функцію preg_match(), яка шукає в заданому тексті subject збіги з шаблоном pattern:

int preg_match (string pattern, string subject [, array matches [, int flags [, int offset]]])

Використання регулярних виразів істотно уповільнює виконання скрипта, тому, якщо є можливість використовувати вбудовані засоби аналізу рядки, то слід їх і використовувати.

Формат визначення шаблонів:

<Роздільник> <шаблон> <роздільник> [<модифікатори>]

Найбільш часто використовувані роздільники "/" ("|", "@").

$Result = preg_match ($pattern, $subject [, $matches]);

Метасимволи

. Будь-який символ, окрім символу переводу рядка.
preg_match('/./', 'PHP 5', $matches); $matches = [0] => P

? або 0, або один збіг
preg_match('/PHP.?5/', 'PHP 5', $matches);

+ Один або більше збігів.
Наприклад, шаблону '/a+b/' будуть відповідати рядки 'ab', "aab", 'aaaaaaab'
preg_match('/a+b/', 'caaabc', $matches);
$matches = [0] => aaab

* Збіги можуть бути, а може їх і не бути
'/de*f/' будуть відповідати рядки 'df', 'def', 'deeeef' і т.д.

Якщо потрібно вказати кількість входжень, використовуємо конструкції

Число, вказане в фігурних дужках говорить про точну кількість входжень {m}, два числа {m, n} вказують діапазон, число з комою вказує на максимальне число входжень {, n} або мінімальне {m}. Ясно, що лічильник починається від нуля.

Тобто '/ tre {1,2} f /' відповідають 'tref' і 'treef', але не 'treeef'

'/fo{2,}ba{,2}r/' будуть відповідати рядка 'foobar', 'fooooooobar' і 'fooobaar', але не 'foobaaar'.

^ Задає вид початку фіксованого підрядка,який перевіряється на входження, так для
'/^abc/' підходить 'abcd' і 'abce', але не підходить 'qabc'.

$ визначає кінець рядка
'/John$/' підходить "Mike, John", але не "John Smith".

[...] Клас відповідних символів.
preg_match('/[0-9]+/', 'PHP is released in 2005',$matches);
$matches = [0] => 2005
Якщо символ ^ стоїть у квадратних дужках, то це заперечення!!!
preg_match('/[^0-9]+/', 'PHP is released in 2005',$matches);
$matches = [0] => PHP is released in

(...) Групування елементів.
preg_match('/([12][0-9])([0-9]{2})/','PHP in 2005",$matches);
Перший елемент масиву є вибраним рядком, що складається з двох груп- спочатку 1 або 2, потім будь-яке число від 0 до 9, потім від 0 до 9 два символи.
$matches = [0] => 2005, [1] => 20, [2] => 05

(?:...) Фрагмент рядка, який не повинен попасти на вихід регулярного виразу.
preg_match('/([A-Za-z]+)(?:ith)/', 'John Smith',$matches);
$matches = [0] => John Smith, [1] => John Sm
Знову дві групи – перша ([A-Za-z]+), куди попав весь рядок і друга -(?:ith), те ж, але без ith.

(?P<имя>...) Визначає іменований вкладений шаблон.
preg_match('/(?P<century>[12][0-9])(?P<year>[0-9]{2})/', 'PHP in 2005',$matches);
$matches = [0] => 2005, [century] => 20, [1] => 20, [year] => 05, [2] => 05

Спеціальні послідовності символів

\?\+*\ [\] \ {\} Екранування символів (\ слеш), які є елементами рядка, а не елементами управління, так замість рядка 4 ** можна записати '/^4 **$/'. Враховуючи, що слеш сам по собі спеціальний символ, який також потрібно екранувати, то для $subject = 'PHP\5'; в залежності від того, які лапки використовуємо, потрібно писати $pattern1 = '/^PHP\\\5$/'; и $pattern2 = "/^PHP\\\\5$/";

\t \n \f \r ASCII 9, 10, 12, 13
\xhh hex 16-ричний код
\ddd 8-ричный код
\d [0-9] цифра
\D [^0-9] не цифра
\s [\t\n\f\r] щось з цих символів (і проміжок)
\S [^\t\n\f\r] не з них
\w Будь-яка буква, цифра, символ підкреслювання
\W Протилежність \w
\b Позиція між суседніми символами \w і \W.
$string = "##Testing123##";
preg_match('/\b.+\b/', $string, $matches);
Так як символ # не входить в \w, то
$matches = [0] => Testing123
\B Протилежніст \b

Функції пошуку

preg_match() повертає true або false, в залежності від результату пошуку.
Наприклад, перевірка, чи є змінна числом

```
$var='12';
preg_match('/^\d+$/', $var);
```

preg_match_all() – повертає всі результати в масив.

Функції заміни

Наступний скрипт міняє теги в квадратних дужках на теги HTML

```
$text= "[B]Hello[/B] world from [B]PHP[/B]";
$newtext = ereg_replace ("\[B\]", "<B>", $text);
$newtext = ereg_replace("\[/B\]", "</B>", $newtext);
```

Другий приклад -всі числа міняємо на X

```
$text= "13hkl32131h";
$newtext = ereg_replace("[[:digit:]]", "X", $text);
```

Функціє розподілу на частини

array preg_split (string pattern, string subject [, int limit [, int flags]])

Повертає масив, що складається з підрядків заданого рядка subject, яка розбита згідно шаблону pattern.

У випадку, якщо параметр limit вказано, функція повертає не більше, ніж limit підрядків. Спеціальне значення limit, рівне -1, має на увазі відсутність обмеження, це дуже корисно для вказівки ще одного опціонального параметра flags.

flags може бути довільною комбінацією наступних прапорів (з'єднання відбувається за допомогою оператора '|'):

PREG_SPLIT_NO_EMPTY

У випадку, якщо цей прапор вказано, функція preg_split () поверне тільки непорожні підрядки.

PREG_SPLIT_OFFSET_CAPTURE

У випадку, якщо цей прапор вказано, для кожного знайденого підрядка, буде вказана його позиція у вихідному рядку. Необхідно пам'ятати, що цей прапор змінює формат даних: кожне входження повертається у вигляді масиву, в нульовому елементі якого міститься знайдений підрядок, а в першому - зрушення.

Контрольні питання.

1. Який результат дії регулярного виразу
   ```
   $string ="aaabbbaaabbbaaa";
   preg_match('/(a.+?b)/', $string, $matches);
   ```
2. Який результат дії регулярного виразу
   ```
   $string ="aAaabBa12bbaAaZab_bbaaa";
   preg_match('/[a-z]+/i', $string, $matches);
   ```
3. Який результат дії регулярного виразу
   ```
   $string ="mama\npapa\nbaby";
   preg_match('/^papa/m', $string, $matches);
   ```
4. Який результат дії регулярного виразу
   ```
   preg_match('/ma.pa/s', $string, $matches);
   ```
5. Який результат дії регулярного виразу
   ```
   $str = 'hypertext language programming';
   $chars = preg_split("/[\s,]+/", $str);
   ```
6. Який результат дії регулярного виразу
   ```
   $str = 'hypertext language programming';
   $chars = preg_split('/ /', $str, -1, PREG_SPLIT_OFFSET_CAPTURE);
   ```

Тема 4: Робота з СУБД

PDO (PHP Data Object)

Це розширення PHP починаючи з версії 5.1, що дозволяє реалізувати єдиний інтерфейс роботи з різними базами даних. PDO реалізує рівень абстракції, при якому робота з БД не залежить від особливостей використовуваної бази даних. Реалізують ці можливості файли, що знаходяться в директорії php/ext, це php_pdo.dll і, якщо ми працюємо з mySql, то php_pdo_mysql.dll, і у файлі конфігурації php.ini, треба зняти коментар з рядка
extension = php_pdo.dll і відповідно, extension = php_pdo_mysql.dll.
Всього можна підключити наступні БД

- MySQL
  ("mysql:host=hostname;dbname=mysql","username", "password")
- SQLite
  ("sqlite:/path/to/database.db")
  ("sqlite::memory:")
- PostgreSQL
  ("pgsql:dbname=pdo;host=hostname", "username", "password")
- Oracle
  ("OCI:dbname=mydatabase;charset=UTF-8","username", "password")
- ODBC
  ("odbc:Driver={Microsoft Access Driver (*.mdb)};
  Dbq=C:\database.mdb;Uid=Admin")
- Firebird
  ("firebird:dbname=hostname:C:\path\to\database.fdb", "username", "password")
- Informix
  ("informix:DSN=InformixDB", "username", "password")
- DBLIB
  ("dblib:host=hostname:port;dbname=mydb","username","password")

Список підключених драйверів можна отримати наступним чином

```
<?php
        echo 'Перелік підключеніх драйверів баз даних <br/>';
        foreach (PDO :: getAvailableDrivers () as $driver) {
        echo $driver. '<br/>';
        }
?>
```

Деякі можливості PDO.

Використання методів роботи з об'єктами дозволяє відокремити модель від бази даних, абстрагуючись від моделі даних. PDO працює з драйвером бази даних досить швидко.
PDO дозволяє вибирати об'єкт із запису БД з полями, назва яких збігається з назвою полів таблиці, при цьому є можливість вибирати не один об'єкт, а масив об'єктів.
Транзакція дозволяє виконувати декілька операцій з базою даних як одну за принципом -або всі разом, або ні одну.

Розглянемо декілька прикладів. Для всіх наведених прикладах реалізовано інтерфейс взаємодії PDO з базою даних MySQL.
Перший приклад– основні дії з СУБД.

```php
<?php
	$host='localhost';
	$login='root';
	$pass='';
	$db_name='users';
	//Підключення СУБД
	$db = new PDO("mysql:host=$host;dbname=$db_name", $login, $pass);
	//INSERT
	/*
		$count = $db->exec("INSERT INTO user(name,email) VALUES ('Alex','alex@mail.com')");
		echo $count;
	*/
	//UPDATE
	/*
		$count = $db->exec("UPDATE user SET email='alex@mail.ua' WHERE name='Alex'");
		echo $count;
	*///Захист від sql-ин'єкцій, будь-який запис екранується та переводиться просто в текстовий рядок
	/*
	$name = $db->quote('Маша');
	$email =$db->quote('mary@mail.com');
	$sql="INSERT INTO user(name,email) VALUES ($name,$email)";// вместо ('$name','$email')
	echo $sql;
	$count = $db->exec($sql);
	echo $count;
	*/
?>
```

Другий приклад – вибірка даних з СУБД

```php
<?php
	$host='localhost';
	$login='root';
	$pass='';
	$db_name='users';
	$db = new PDO("mysql:host=$host;dbname=$db_name", $login, $pass);
	$sql="SELECT * FROM user";
	/* //Пряма вибірка з бази
	foreach ($db->query($sql) as $row){
		echo $row['name'] . ' - ' . $row['email'] . '<br/>';
	}
	*/ //Перетворення в асоціативний масив
	$stmt=$db->query($sql);
	$result=$stmt->fetch(PDO::FETCH_ASSOC);
	//while($result=$stmt->fetch(PDO::FETCH_ASSOC))
	foreach($result as $key=>$val){
		echo $key . ' - '. $val . '<br/>';
	}
?>
```

Наступний приклад – обробка виключень та помилок.

```
<?php
	$host='localhost';
	$login='root';
	$pass='';
	$db_name='users';
	try{
		$db = new PDO("mysql:host=$host;dbname=$db_name", $login, $pass);
		echo "З’єднання з БД <br/>";
		$db-
		>setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION)
		;
		//Обробка виключень
		//$db-
		>setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_WARNING);
		//Використовуємо попередження
		$db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_SILENT);
		//Вся інформація, яка пов'язана з базою прибирається, інша
		//виводиться (наприклад, помилки операторів)
		$sql="SELECT username FROM user";  //вказано неіснуюче поле
		foreach($db->query($sql) as $row){
			echo $row['name'] . ' - '. $row['email'] . '<br/>';
		}
		$db=null;
	}catch(PDOException $e){
		echo "Обробка виключень<br/>";
		echo $e->getCode(). " : ". $e->getMessage();
	}
	echo "<br/>";
	//Можно так
	//echo $db->errorCode();
	//print_r($db->errorInfo());
?>
```

У PDO є можливість обробки вхідних даних в будь-якому з трьох варіантів – як індексований масив, як асоціативний масив і як об’єкт, така собі можливість для лобурів (lazy)

```
<?php
	$host='localhost';
	$login='root';
	$pass='';
	$db_name='users';
	$db = new PDO("mysql:host=$host;dbname=$db_name", $login, $pass);
	$sql="SELECT * FROM user";
	$stmt=$db->query($sql);
	while($result=$stmt->fetch(PDO::FETCH_LAZY)){
		echo $result[0] . '<br/>';
		echo $result['name'] . '<br/>';
		echo $result->email . '<br/>';
	}
?>
```

Контрольні питання.

1. Чим відрізняється підготовлений запит від SQL-запиту?
2. Що виконує функція execute();
3. Як визвати збережену процедуру з вихідними параметрами?
4. Як визвати збережену процедуру з вихідними та вхідними параметрами?
5. Які можливості обробки даних у lazy режимі?

Тема 5: Об’єктно орієнтований PHP. Паттерни проектування.

Клас - контейнер для логічно пов'язаних даних і функцій. Фактично клас - це тип даних. Приклад оголошення класу:

```
class Article
{
        // Тіло класу
}
```

Імена класів в PHP прийнято писати з великої літери.

Об'єкт - сукупність конкретних даних і функцій для їх обробки. Фактично об'єкт - це змінна, тип даних якої задається відповідним класом, тобто екземпляр класу. Для оголошення об'єкта використовується ключове слово new.

Приклад оголошення об'єкта:

```
$a = new Article ();
```

Поля класу - дані, які зберігає клас. Також поля класу називають властивостями класу.

```
class Article
{
        // Нижче представлені три властивості класу
        // Article: id, title, content
        var $id;
        var $title;
        var $content;
}
```

Звернення до полів об'єкта відбувається за допомогою оператора «->». Приклад:

```
$a = new Article (); // Об'єкт $a це екземпляр класу
echo $a-> id;
```

Метод - функція, оголошена всередині класу. Метод має доступ до всіх полів класу. Для звернення до власних полях всередині методів класу використовується ключове слово this.

```
class Article
{
  // Оголошення полів класу
  // ...
  // Функція для виведення статті
  function view () {
    echo "<h1> $this-> title </ h1> <p> $this-> content </ p>";
  }
}
```

Доступ до методів реалізується аналогічно доступу до полів, наприклад,

```
$a-> view ();
```

Знаючи структуру класу, можна використовувати властивості класу ззовні, наприклад, якщо викликати функцію

```
function viewArticle ($article)
{
        echo "<h1>$article-> title</h1><p> $article->content </p>";
}
$A = new Article();
viewArticle ($a);
```

У розглянутому прикладі ми створюємо порожній екземпляр класу, який наповнюємо конкретними полями. Цього можна уникнути, якщо використовувати конструктор класів.

Конструктор - метод, який викликається при створенні екземпляра даного класу. Один з варіантів оголошення конструктора - функція з назвою, що збігається з ім'ям класу. Приклад:

```
class Article
{
  ... // Оголошення полів класу
  function Article ($id, $title, $content)
  {
    $This-> id = $id;
    $This-> title = $title;
    $This-> content = $content;
  }
  ... // Оголошення методів класу
}
```

Виклик конструктора проводиться наступним чином

```
$a = new Article (1, 'Тема', 'Текст статті');
```

Три ключових поняття ООП

Інкапсуляція - властивість мови програмування, що дозволяє об'єднати дані і код в об'єкт і приховати реалізацію об'єкта від користувача (прикладного програміста). При цьому користувачеві надається тільки специфікація (інтерфейс) об'єкта.

Спадкування - один з найважливіших механізмів об'єктно-орієнтованого програмування, що дозволяє описати новий клас на основі вже існуючого (батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом.

Для спадкування класів в PHP використовується ключове слово extends. Приклад:

```
class Article
{
  ... // Вміст класу
}
class NewsArticle extends Article
{
  ... // Вміст класу
}
```

Класи-спадкоємці можуть мати свої методи, а можуть і перевизначати методи класів-батьків. Приклад:

```
class Article
{
  ...
```

```
  // Функція для виведення статті
 function view ()
 {
   echo "<h1> $this-> title </ h1> <p> $this-> content </ p>";
 }
}
 class NewsArticle extends Article
{
  ...
   // Функція для виведення статті
 function view ()
 {
   echo "<h1> $this-> title </ h1> <span style = 'color: red'>".
       strftime ('% d.% m.% y', $this-> datetime).
       "<B> Новина </ b> </ span> <p> $this-> content </ p>";
 }
}
```

У даному прикладі дочірній клас NewsArticle перевизначає метод view батьківського класу Article.

Поліморфізм - взаємозамінність об'єктів з однаковим інтерфейсом.
Мова програмування підтримує поліморфізм, якщо класи з однаковою специфікацією можуть мати різну реалізацію - наприклад, реалізація класу може бути змінена в процесі успадкування.

```
class A
{
  function Test () {echo 'Це клас A <br />'; }
  function Call () {$this-> Test (); }
}
 class B extends A
{
  function Test () {echo 'Це клас B <br />'; }
}
$A = new A();
$B = new B();
$A-> Call(); // Виводить: «Це клас A»
$B-> Test(); // Виводить: «Це клас B»
$B-> Call(); // Виводить: «Це клас B»
```

Специфікатори (модифікатори) доступу Private, Public, Protected
Метою специфікаторів є обмеження доступу до полів класу.
Модифікатор public дозволяє звертатися до властивостей і методів звідусіль, тобто ніяк не обмежує, як усередині класу, так і поза ним.
Модифікатор private дозволяє звертатися до властивостей і методів тільки всередині поточного класу.
Модифікатор protected дозволяє звертатися до властивостей і методів тільки поточного класу і класу, який успадковує властивості і методи поточного класу.

```
class Article
{
  private $id;
```

```
  protected $content;
  public $title;
    ...
// Функція для виведення статті - має доступ до всіх полів класу
  function view () {
echo "<h1> $this-> id - $this-> title </ h1> <p> $this-> content </ p>";
  }
  function printId () {
echo $this-> id;
   }
}
echo $a-> id;
// Помилка: модифікатор private не дозволяє отримати доступ
$A-> printId ();
// Ok доступ усередині класу
echo $a-> content;
// Помилка: модифікатор protected не дозволяє отримати доступ
echo $a-> title;
// Все ок: модифікатор public дозволяє отримати доступ

class NewsArticle extends Article
{
  ...
// Функція для виведення статті - має доступ до полів content // і title, але не id!
  function view ()
  {
    echo "<h1> $this-> title </ h1> <p> $this-> content </ p>";
  }
}
```

Аналогічна ситуація з методами класу. Так, якщо записати private function printId () і викликати не в класі, то буде помилка.

Уніфікований конструктор __construct ()

Конструктори в класах- батьках не викликаються автоматично. Щоб викликати конструктор, оголошений в батьківському класі, слід звернутися до методу parent :: __ construct (). Зауважимо, що подвійне підкреслення - це перевизначення стандартної функції

```
class Article
{
...
  function __construct ($id, $title, $content)
  {
    $This-> id = $id;
    $This-> title = $title;
    $This-> content = $content;
  }
...
}
```

Деструктори __destruct ()

Коли звільняється останнє посилання на об'єкт, перед вивільненням пам'яті, займаної цим об'єктом, викликається метод __destruct (), що не приймає параметрів.

```
class Article
{
 ...
 function __destruct ()
 {
  echo "стаття знищується <br/>";
 }
 ...
}
```

Клонування об'єктів

Копія об'єкта створюється з використанням оператора clone. Для вказівки власної логіки поведінки об'єктів при клонуванні необхідно перевизначити функцію __clone (), яка буде викликана після копіювання всіх полів вихідного об'єкта в новий.

```
class Article
{
 ...
// Функція клонування збільшує значення поля id нового об'єкта на одиницю
 function __clone ()
 {
  $this-> id = $this-> id + 1;
 }
 ...
}
$a = new Article (1, 'Тема', 'Зміст');
$b = clone $a;
$a-> view (); // Виведе статтю з id == 1
$b-> view (); // Виведе стаття з id == 2
```

Тобто, $a і $b вказують на різні об'єкти. Якщо замість $b = clone $a; написати $b = $a; то ми отримаємо два посилання на один і той же об'єкт, при цьому навіть деструктор відпрацьовує тільки один раз.

Статичні члени класів

Визначення класів можуть включати статичні члени (як властивості, так і методи), доступ до яких здійснюється через клас. Тобто, скільки б ми не створювали екземплярів класу, статичний елемент класу буде один і той же. Статичні методи можуть працювати тільки зі статичними членами класу. Звернення до статичних властивостями всередині методів класу відбувається за допомогою ключового слова self. Для виклику статичних методів класу використовується наступний синтаксис:

<Ім'я_класу> :: <імя_метода> (<параметри>).

```
class Singleton
{
    static private $instance = NULL;
    ...
    static public function getInstance ()
{
```

```
            if (self :: $instance == NULL)
        {
                self :: $instance = new Singleton ();
            }
            return self :: $instance;
        }
        }
        $Instance = Singleton :: getInstance ();
```

Абстрактні класи та методи

Починаючи з PHP 5 підтримується визначення абстрактних класів і методів. Створювати екземпляр класу, який був оголошений абстрактним, не можна. Клас, в якому оголошено хоча б один абстрактний метод, повинен також бути оголошений абстрактним. Методи, оголошені як абстрактні, мають описовий характер і не можуть включати будь-якої функціонал. Класи і методи можуть бути оголошені як абстрактні за допомогою використання ключового слова abstract. Зазвичай абстрактні класи створюються для того, щоб їх дочірні класи перевизначили відсутні методи.

```
        abstract class Article
        {
          ...
          abstract function intro (); // Опис відсутній
          function view ()
          {
            $This-> intro ();
        echo "<h1> $this-> id - $this-> title </ h1> <p> $this-> content </ p>";
          }
        }
        class NewsArticle extends Article
        {
          ...
          function intro ()
          {
            echo '<span style = "color: red">'.
                strftime ('% d.% m.% y', $this-> datetime).
                '<B> Новина </ b> </ span>';
          }
        }
```

Тобто, якщо ми створимо екземпляр класу Article

```
        $A = new Article (1, 'Тема', 'Зміст');
```

то буде помилка.

Вказівка класу як типу

Визначення функції можуть включати вказівку типу класу, переданого як параметр. Якщо функція буде викликана з неправильним типом, відбудеться помилка.

```
        class ArticleList
        {
          ...
          function add (Article $article)
```

```
  {
    $This-> alist [] = $article;
  }
  ...
}
$A = new Article (...);
$ArtList = new ArticleList ();
$ArtList-> add ($a);
$ArtList-> add (3);// Помилка: число не є екземпляром класу Article
```

Можна передавати екземпляри дочірніх класів, так якщо

```
class NewsArticle extends Article
{...}
```

і

```
$b = new NewsArticle (...);
```

то можна використовувати

```
$ArtList-> add ($b);
```

Розіменування об'єктів, які повертаються методами

Розіменування дозволяє використовувати при визначенні методів або властивостей об'єкти інших класів, наприклад,

```
$this-> source = new SourceInfo ($link);
```

і при цьому можна повертати екземпляр цього класу, наприклад,

```
function getSource ()
{
return $this-> source;
}
```

і для екземпляра класу $a можна записати

```
$a-> getSource () -> viewLink ();
```

Робота з класом як з масивом

Починаючи з PHP 5 з'явилася можливість працювати з полями об'єктів, як з елементами масиву. За замовчуванням, в ітерації будуть брати участь всі властивості, оголошені як public.

```
$a = new Article (1, 'Тема', 'Зміст');
foreach ($a as $field)
  echo $field. '<br />';
```

Винятки

PHP додає парадигму обробки виключень, вводячи конструкцію try / throw / catch. Для розробки власних класів винятків необхідно наслідувати клас Exception.

```
class SQLException extends Exception
{
   public $problem;
   function __construct ($problem)
  {
      $This-> problem = $problem;
   }
}
try
{
```

```
  ...
  throw new SQLException('Не можу встановити з'єднання з БД');
  ...
}
catch (SQLException $e)
{
    echo $e-> problem;
}
catch (Exception $e)
{
   echo $e-> getMessage ();
}
```

Перевантаження властивостей класів

Звернення до властивостей об'єкта можуть бути перевантажені з використанням методів
__get і __set. Ці методи будуть спрацьовувати тільки в тому випадку, якщо об'єкт або успадковані об'єкт не містять властивості, до якого здійснюється доступ.

void __set (string ім'я, mixed значення)

Функція __set приймає два параметри. Перший - ім'я поля, яке ми намагаємося встановити. Другий - значення.

void __get (mixed ім'я)

Функція __get приймає один параметр, ім'я запитуваної поля.

```
class Article
{
  ...
  private $links = array ();
  // Установка властивості класу
  function __set ($field, $val)
  {
    $This-> links [$field] = $val;
  }
  // Отримання властивості класу
  function __get ($field)
  {
return '<a href="'.$this-> links [$field].' "> '. $field.' </a> ';
  }
  ...
}
$A = new Article (1, 'Тема', 'Зміст');
$A-> pzs = 'http://pzs.dstu.dp.ua'; // Поле не існує
$A-> dstu = 'http:// dstu.dp.ua '; // Поле не існує
$A -> {'Кафедра ПЗС'} = 'http://pzs.dstu.dp.ua';
```

На цьому етапі викликається функція __set (), яка виконує деякі дії (у нашому випадку сформує масив links)

```
echo $a-> pzs. '<br />';
echo $a-> dstu. '<br />';
echo $a -> {'Кафедра ПЗС'};
```

на цьому етапі викликається функція __get (), яка виконує дії з результатом роботи __set () по ключу $field.

Перевантаження методів

Виклики методів можуть бути перевантажені з використанням методу __call. Цей метод буде спрацьовувати тільки в тому випадку, якщо об'єкт або успадковані об'єкт не містять методу, до якого здійснюється доступ.

mixed __call (string ім'я, array аргументи)

В якості першого параметра метод __call приймає ім'я запитуваного методу. В якості другого параметра - масив переданих аргументів.

```
class Article
{
 ...
  // Виклик невідомого методу
 function __call ($method, $params)
 {
  switch ($method)
  {
    case 'print':
    case 'echo':
    case 'dump':
    case 'list':
      $This-> view ();
    break;
    default:
      die ('такого методу не існує');
    break;
  }
 }
 function view ()
 {
  echo "<h1> $this-> id - $this-> title </ h1> <p> $this-> content </ p>";
 }
}
$A = new Article (1, 'Тема', 'Зміст');
$A-> view ();
$A-> echo ();
$A-> print ();
$A-> list ();
```

Інтерфейси

Інтерфейси об'єктів дозволяють програмісту створювати код, який вказує, які методи повинен включати клас, без необхідності описування їх функціоналу.

Інтерфейси оголошуються так само, як і звичайні класи, але з використанням ключового слова "interface"; тіла методів інтерфейсів повинні бути порожніми. Для включення інтерфейсу в клас використовується ключове слово "implements" і треба описати функціонал методів, перерахованих у включається інтерфейсі. Якщо це потрібно, класи можуть включати більше одного інтерфейсу шляхом їх перерахування через проміжок. Якщо клас включає будь-якої інтерфейс і не описує функціонал всіх методів цього інтерфейсу, виконання коду з використанням такого класу завершиться фатальною помилкою, що повідомляє, які саме методи не були описані.

interface Article

```
{
  public function view ();
  public function setTitle ($title);
  public function getTitle ();
}
class Article implements IArticle
{
  ...
  function setTitle ($title)
  {
    $This-> title = $title;
  }
  function getTitle ()
  {
    return $this-> title;
  }
  function view ()
  {
    echo "<h1>$this->id - $this-> title </h1><p> $this-> content";
  }
}
```

Оператор instanceof

Оператор instanceof перевіряє, чи є об'єкт екземпляром заявленого класу, екземпляром класу спадкоємця, екземпляром класу, що включає заявлений інтерфейс.

```
interface IArticle
{
  public function view ();
}
...
foreach ($articles as $art)
{
  if ($art instanceof IArticle)
    $Art-> view ();
}
```

Методи final

Ключове слово final дозволяє позначати методи, щоб дочірній клас не міг перевантажити їх.

```
class Article
{
  ...
  final function view ()
  {
    echo "<h1>$this->id - $this->title </h1><p>$this->content";
  }
}
class NewsArticle extends Article
{
```

```
  ...
  // Помилка: метод view перевантажувати не можна!
  function view ()
  {
    echo "<h1> Новина: $this-> id - $this-> title </ h1>
<P> $this-> content </ p> ";
  }
}
```

Класи final

Після оголошення класу final він не може бути успадкований.

```
final class Article
{
  ...
}
class NewsArticle extends Article
{
  ...
}
// Помилка: у класу Article не може бути нащадків!
```

Контрольні питання.

1. Для чого призначений Інтерфейс Iterator (SPL)?
2. Який Інтерфейс SPL реалізує серіалізацію?
3. Які функції паттерну Observer?
4. Чим відрізняється паттерн «Фабрика» від паттерну «Фабричний метод»?
5. Що породжує паттерн «Одиночка»?

Частина 2. XML –технології

Тема 1. XML

Документ може містити

- Дані
- Структуру
- Рівень представлення

XML працює з даними, які є текстом.

Для кожної мови існують рівні:

- **Синтаксис** - правила запису елементів розмітки (в html теги можуть писатися як маленькими, так і великими літерами, для парних тегів, якщо є що відкриваючий, то повинен бути і закриваючий тег)
- **Граматика** - відношення елементів розмітки один до одного (у html- в елементі table всередині тега tr стоять теги td, у мові С блок виділяється фігурними дужками)
- **Семантика** - смислові значення елементів розмітки (наприклад, теги <b> і , результат їх застосування не відрізняється, але <b> - жирний, а - важливий або <p> і , перший - параграф, тобто відступ, другий - блоковий елемент, розділ). Відмінності - не можна всередині параграфа зробити ще один параграф, а всередині блоку зробити інший, елементарно.

У XML є синтаксис (теги можуть бути вкладені, але не можуть перетинатися), тобто правила запису. І без додаткової інформації немає ні граматики, ні семантики. Наприклад, якщо є документ

```
<?xml version=“1.0” encoding=”windows-1251”?>
<!-- Приклад XML размітки -->
<pricelist>
        <book id=”1”>
                <title>XML и IE5</title>
                <author> Алекс Гомер </author>
                <price currency =”UAH”>200</price>
                <exists/>
        </book>
</pricelist>
```

Без додаткової інформації немає ніякої впевненості, що <author> повинен бути вкладений в <book>, тобто є проблеми з граматикою.

Крім того, чи повинен елемент <author> містити інформацію саме про автора книги. Ні, звичайно, це може бути що завгодно, наприклад, автор критичних зауважень.

Тобто жоден тег нічого не означає, в чистому виді у XML немає ні граматики, ні семантики. АЛЕ. Але сенс з'являється, як тільки з'являється контекст, тобто в роботі над конкретним документом - нехай тег <author> позначає автора цієї книги. Тоді з'являється і граматика - автор цієї книги. Таким чином, XML нескінченно

розсширяєма мову. Опис граматики проводиться використанням DTD і XML-shema.

XML-чутливий до регістру. Допускається використання символів, в тому числі і кирилиці, цифр, символу підкреслення, крапка та ін. Що не можна використовувати – пусті проміжки, ім'я елемента завжди починається з літери (хоча деякі аналізатори цього не вимагають), а також використовувати три заборонених символи '>', '<', '&'. Всі теги парні. Одиночних немає в принципі, але. Якщо вміст тега порожній, то можна використовувати скорочення, так замість

<exists></exists> можно писати <exists />

(Перед слешем проміжок!). Аналогічно, для html тег
. Значення атрибутів завжди беруться в подвійні лапки. У будь-якого документа завжди буде один тег - кореневий. Все, що починається на! є декларація (оголошення).

За замовчуванням, все з чого складається XML- документ - парсується (аналізовані дані), тобто, PCDATA дані.

У випадку, якщо не слід проводити парсінг, використовують простий текст - CDATA.

```
<?xml version="1.0" encoding="windows-1251"?>
        <data>
                <desc>
                        <![CDATA[
                        a>b & c
                        ]]>
                </desc>
        </data>
```

Але якщо потрібно використовувати розмітку представлення (html), то

```
<?xml version="1.0" encoding="windows-1251"?>
<!-- Пример XML разметки -->
<pricelist>
        <book id="1">
                <title>XML и IE5</title>
                <author> Алекс Гомер </author>
                <price currency ="UAH">200</price>
                <description>
                        <![CDATA[
                        Книга описує представлення XML <br/>
                        у браузері IE5
                        ]]>
                </description>
                <exists />
        </book>
</pricelist>
```

Будь-який документ складається з двох блоків - з необов'язкового прологу та обов'язкових даних.

Пролог - це декларація самого XML і процесингові інструкції, а також посилання на граматику, наприклад

<?xml version=”1.0” ?>

<!DOCTYPE employees SYSTEM “employees.dtd”>

Самі дані завжди знаходяться всередині кореневого елемента

<employees>
    <name>Alexander</name>
</employees>
Без декларації XML все чудово працює, поки використовується тільки латиниця. Тобто без вказівки на кодуву сторінку можна використовувати тільки латиницю. Будь який інший символ вимагає вказівки кодової сторінки.

Простори імен.

Простори імен усувають конфлікти елементів. Оголошення простору імен проводиться за допомогою Uniform Resource Identifier (URI). URI може бути URL або URN

URL: http: //www/w3.org/1999/XSL/Transform

URN: urn: schemas-microsoft-com: xml-data

У випадку роботи з великими проектами, коли є ймовірність, що в проекті можуть попастися елементи з одним ім'ям, слід використовувати простори імен, в такому разі елементи з однією назвою лежатимуть в «паралельних всесвітах».

Причому, аналізатор не заходитиме на цю адресу. Це просто унікальна рядок. Тому, вважаючи, що всі веб-сторінки мають унікальну адресу, їх можна використовувати в якості покажчика простору імен. Вказівка на стандарт w3.org це правило хорошого тону.

Є два способи роботи з простором імен.

• Глобальне або пряме оголошення - використання атрибуту xmlns.

<payment xmlns=”http://mysite/xml/pay”>
    <customer>123</customer>
</payment>
Тут елемент payment з простору імен "http://mysite/xml /pay". У цьому випадку прив'язка до простору імен проводиться для всіх вкладених елементів, якщо не передбачено інше.

Таким чином, вказівка простору імен у кореня призводить до того, що всі елементи будуть прив'язані до цього простору імен, що не заважає конкретні елементи прив'язувати до іншого простору.

Наприклад, потрібно зробити форматований документ, але коштів форматування у xml немає, зате є у html. Крім того, є xhtml, той же html, але з тегами, оформленими за правилами xml.

<?xml version=“1.0” encoding=”windows-1251”?>

```
<pricelist>
        <book id=”1”>
                <title>XML  и IE5</title>
                <description xmlns:”http://www.w3.org/1999/xhtml”>
                        Книга описує застосування XML <br/>
                        в браузері <b>IE5</b>
                </description>
                <exists />
        </book>
</pricelist>
```

• Інший метод використання простору імен зводиться до використання префікса.

У цьому випадку простір імен застосовується лише до тих тегів, де є пряма декларація (прив'язка) даного простору імен.

```
<a:payment xmlns:a=”http://mysite/xml/pay”>
        <customer>123</customer>
</a:payment>
```

Тому, хоча в payment і прописано простір імен, його все одно потрібно декларувати, тобто записувати a: payment.

Тому краще простору визначати в кореневому елементі

```
<root xmlns:html=”http://www.w3.org/1999/xhtml”

        xmlns:a=”http://mysite/xml/pay”>
```

Для того, щоб не було конфлікту, доцільно вказати відразу загальний простір імен, в якому лежать всі теги, не прив'язані до конкретних інших просторів

```
<root xmlns=”http://mysite/xml/pay”
    xmlns:book=”http://mysite/xml/book”
        xmlns:html=”http://www.w3.org/1999/xhtml”>
        <book:title>XML  и IE5</book:title>
                <book:description>
                        Книга описує застосування XML < html:br />
                        в браузері <html:b>IE5</html:b>
                </book:description>
                <exists />
```

По простору можна позначати не тільки елементи, але й атрибути.

За замовчуванням атрибут належить тому простору, якому належить елемент. Але реально можна вказати значення атрибута з простору відмінного від простору елемента

```
<a:lesson html:id=”1”>
```

У класичному XML граматики і семантики (смисловий опис) нема, якщо семантика виникає в процесі створення документа, а граматика (правила і ставлення елементів цього документа) складається заздалегідь.

Перевірка документа (структура, дані) на етапі його завантаження (валідація) є необхідним елементом роботи з XML-документом. Набір описаних правил, які

повинні бути граматики, дозволяє автоматизувати процес валідації. Для цього існують два інструменти DTD -Document Type Definition і XML-Schema.

Контрольні питання.

1. На якому етапі у XML виникає семантика?
2. Чи є HTML підмножиною XML?
3. В чому сенс використання простору імен?
4. Що є в наявності у HTML- синтаксис, граматика, семантика?

Тема2. Валідація даних.

DTD -Document Type Definition

Яка мета - опис

- Які елементи присутні в документі
- Повторення елементів
- Які атрибути можуть бути у елементів
- Які атрибути обов'язкові
- Які сутності можуть застосовуватися.

В html не можна використовувати '<', '>', '&' і деякі інші символи, замість них використовуються константи (сутності) & lt; & gt; & amp; & nbsp; © та інші. Але з їх використанням в XML є проблеми

<html:div>

< > &   ©

</html:div>

Не призведе до необхідного результату, так як простір імен html: відноситься тільки до тегу div, а не до вмісту. У XML розширюється все, в тому числі і сутності. І це все описується в граматиці.

```
<!ELEMENT pricelist (book+)>
<!ELEMENT  book (title, author+, price, description?)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT author(#PCDATA)>
<!ELEMENT price(#PCDATA)>
<!ELEMENT description(#PCDATA)>
<!ATTLIST price currency CDATA #IMPLIED>
```

Умовно кажучи '<!' Декларація 'Нехай буде ...'

Нехай буде елемент pricelist, у якого є елементи book

Нехай буде елемент book, у нього будуть title, author, price, description

Нехай буде елемент title, у нього всередині текст ...

Оголошення DTD

- ELEMENT - визначення елемента
- ATTLIST - визначення атрибута

- ENTITY - визначення сутності

Модифікатори

- * Нуль або багато
- ? нуль або один
- + один або багато

У дужках через кому перераховують дочірні елементи. У перерахуванні можна використовувати альтернативи

(Price|sample|chargeacct) АБО (ціна або приклад або платіжний рахунок)

Якщо елемент має бути порожнім, то пишемо EMPTY

<! ELEMENT warning EMPTY>

Якщо текст аналізований (парсований), то вказуємо #PCDATA, якщо непарсований, то CDATA.

Атрибути можуть визначатися для будь-якого елементу з використанням декларації ATTLIST. Відразу можна визначити декілька атрибутів одного і того ж елемента із зазначенням #REQUIRED –об’язковий або #IMPLIED необов'язковий.

```
<!ATTLIST price
currency CDATA #REQUIRED
symbol CDATA #IMPLIED>
<price currency =”USD” symbol=”$”>125.25</price>
```

Таким чином, у pricelist не менше одного елемента book. У book один елемент title, один або більше author, один price, може бути один (а може і не бути) description.

Завантажити DTD можна декларативно (використовуючи DOCTYPE) - прямо в документі XML

```
<?xml version="1.0" encoding="windows-1251"?>
<!DOCTYPE pricelist[
<!ELEMENT pricelist (book+)>
<!ELEMENT  book (title, author+, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA #IMPLIED>
]>
<pricelist>
<book>
                        <title>XML  и IE5</title>
                        <author> Алекс Гомер </author>
                        <price currency ="UAH">200</price>
</book>
</pricelist>
```

Такий документ називається самодостатній. У старих версіях це зазначалося

```
<?xml version="1.0" standalone=”yes” encoding="windows-1251"?>
```

Зараз даний запис не використовується, як втім, і самі самодостатні документи.

Можна використовувати зовнішні DTD - типу PUBLIC це стандартні загальновідомі з посиланням на ресурс, де викладений цей стандарт і кожен його може подивитися і типу SYSTEM - ніби як моє, з посиланням на цей DTD. DTD завжди визначає кореневий елемент документа.

<!DOCTYPE pricelist SYSTEM ”books.dtd”>

Тепер, якщо структура документа не відповідає DTD, то документ не пройде валідацію і браузер не зможе обробити цей документ.

Введення нових сутностей (констант)

<!ENTITY wil “Williams”>

<!ENTITY ap “A-Press”>

Приклад використання

<publisher>≈</publisher>

Дає можливість підставляти значення цієї константи.

<!ENTITY company “Рога и копыта”>
<TEXT>Компанія & company; завжди до ваших послуг! </ TEXT>

Головне призначення DTD - валідація (робота з граматикою), тобто, перевірка документа на відповідність стандарту (а цей стандарт можна писати самі).

XML-схеми

У DTD є кілька проблем, які суттєво обмежують його використання і в кінці-кінців призвели до того, що DTD практично повністю поступився своїми позиціями новим методом опису граматики - XML- схемами.

У DTD проблеми з кодуванням, немає можливості використовувати нелатинських алфавіт для опису тегів, немає перевірки типів даних, не можна поставити документу два і більше DTD описів, що актуально для складених документів.

XML-Schema описує

- Словник (назви елементів і атрибутів)
- Модель змісту (відносини між елементами і атрибутами і їх структура)
- Типи даних.

Всі елементи схеми завжди належать простору імен http://www.w3.org/2001/XMLSchema. Як правило, цей простір імен використовують з префіксом xs або xsd, хоча ніякої ролі це не грає, просто умовність.

Приклад.

<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”> <xs:element name=”країна”	<країна xmlns:xsi= ”http://www.w3.org/2001/XMLSchema-instance” xsi:noNamespaceSchemaLocation=

type=”Страна”/> <xs:complexType name=”Країна”> <xs:sequence> <xs:element name=”назва” type=”xs:string”/> <xs:element name=”населення” type=”xs:decimal”/> </xs:sequence> </xs:complexType> </xs:schema>	”country.xsd”> <назва>Франція</назва> <населення>59.7</населення> </країна>

XMLSchema являє собою XML-документ з кореневим елементом

<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”>

Тому всі елементи XMLSchema повинні записуватися через префікс цього простору, в даному випадку, через xs:

XMLSchema являє собою набір елементів, які описуються ключовим словом «element», далі вказується ім'я цього елемента name = "...". Після цього вказуємо тип цього елемента, він може бути простим (simpleType, базового типу) або складовим (complexType, по суті, це структура, що складається з послідовності «sequence» елементів). Часто тип елемента описується при визначенні цього елемента. Зауважимо, що, так як в схемах є спадкування, то типи даних можуть розширятися.

Відповідність схеми документу

XMLSchema
<?xml version=”1.0”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
targetNameSpace=”http://www.camera.org”
xmlns=”http://www.camera.org”
xmlns:nikon=”http://www.nikon.org”
xmlns:olympus=”http://www.olympus.org”
xmlns:pentax=”http://www.pentax.org”
elementFormDefault=”unqualified”>

XMLDocument

<?xml version=”1.0”?>
<my:camera xmlns:my=”http://www.camera.org”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:schemaLocation=”http://www.camera.org/Camera.xsd”>

Якщо схема не є стандартом, то її можна вказати за допомогою елемента schemaLocation, але так як це елемент простору імен XMLSchema, то потрібно це вказати за допомогою відповідного префікса

xsi: schemaLocation,

де префікс вказує на стандарт XMLSchema

xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"

Якщо прив'язку робимо без вказівки простору імен, то використовуємо конструкцію

noNamespaceSchemaLocation

XML-Schema мають досить багато базових типів даних, серед яких Int, Integer, Short, Long, Decimal, Double, Float, string, Name, AnyURI, base64Binary, кілька типів дати, часу - Date, DateTime, Time, опис сутностей ENTITIES (XML-сутності), ENTITY (XML-сутність) і багато інших.

При цьому, багато з них дублюються, так, наприклад, Int (спадщина C) і Integer - цілі числа, нічим один від одного не відрізняються. Name - та ж рядок (string) але з великої літери. Дата, час завжди вказується від більшого до меншого, Data (2014-05-17), Time (14: 05: 00.000), при необхідності можна окремо отримати, наприклад, поточний рік gYear (2015) або місяць і день gMonthDay (-05 -17) і багато іншого. Як правило, кореневий елемент має complexType, тобто, до складу якого входить послідовність тегів (елементів).

Самостійно створіть анкету - ПІБ, група, рік народження, телефон.

Можна вводити свій тип даних, наприклад, введемо "PersonType"

```
<?xml version="1.0" encoding="windows-1251"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="client" type="PersonType"/>
              <xs:element name="clerk" type="PersonType"/>
            </xs:sequence>
          <xs:complexType>
        </xs:element>
          <xs:complexType name ="PersonType">
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="birthdate" type="xs:date"/>
            </xs:sequence>
          <xs:complexType>
</xs:schema>
```

Досить часто виникає необхідність зробити повторення, для нашого прикладу один клерк може обслуговувати декілька клієнтів. У DTD для цієї мети використовувалися модифікатори. У схемах для цієї мети використовуються атрибути minOccurs і maxOccurs, що вказують мінімальне і максимальне число входжень, при цьому від minOccurs = "0" до maxOccurs = "unbounded" - не обмежена. Наприклад,

```
<xs:element name="client" type="PersonType"
  minOccurs=”1”  maxOccurs=”unbounded”/>
```

в цьому випадку кількість клієнтів на одного клерка від одного до необмеженого.

Об'єднувач «sequence» не є обов'язковим, може бути «один з декількох»

```
<xs:complexType name="paraType" mixed="true">
 <xs:choice minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="emph"/>
  <xs:element ref="strong"/>
 </xs:choice>
 <xs:attribute name="version" type="xs:number"/>
</xs:complexType>
```

і «кожній із зазначених»

```
<xs:element ref="egg" minOccurs="12" maxOccurs="12"/>
<xs:group ref="omelette" minOccurs="0"/>
<xs:any maxOccurs="unbounded"/>
```

Можна працювати і з атрибутами, зауважуючи, що атрибут є приналежність типу, для цього використовуємо ключове слово attribute з трьома обов'язковими полями name, type, use (режим використання-required обов'язково, optimal може бути).

```
<xsd:complexType name=”address”>
<xsd:sequence>
  <xsd:element name=”name” type=”xsd:string”/>
    <xsd:element name=”street” type=”xsd:string”/>
    <xsd:element name=”city” type=”xsd:string”/>
</xsd:sequence>
<xsd:attribute name=”phone” type=”xsd:srting” use=”optimal”/>
</xsd:complexType>
```

При цьому опис атрибута входить в об'єднувач «sequence», але після створення елемента.

Якщо, незважаючи на те, що елементи належать одному типу, між ними є відмінності

```
<client>
    <name>Client</name>
    <birthdate>2015-05-17</birthdate>
    <passport>string</passport>
</client>
<clerk>
     <name>Clerk</name>
    <birthdate>2015-05-17</birthdate>
    <id>ID</id>
</clerk>
```

В даному випадку це id і passport. Як бути в цьому випадку, коли частина типу співпадає, а частина відрізняється? Можна створити групу елементів.

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
    <xs:element name="client" minOccurs="1"  maxOccurs="unbounded">
       <xs:complexType>
```

```
                <xs:sequence>
                <xs:group ref="PersonGroup">
                <xs:element name="passport" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
          </xs:element>
          <xs:element name="clerk">
               <xs:complexType>
                  <xs:sequence>
                  <xs:group ref="PersonGroup">
                  <xs:element name="id" type="xs:integer"/>
                 </xs:sequence>
              </xs:complexType>
          </xs:element>
          </xs:sequence>
      </xs:complexType>
</xs:element>
    <xs:group name ="PersonGroup">
          <xs:sequence>
             <xs:element name="name" type="xs:string"/>
              <xs:element name="birthdate" type="xs:date"/>
           </xs:sequence>
    </xs:group>
```

І приєднувати цю групу, створюючи необхідний тип даних.

Можна це завдання вирішити інакше, використовуючи розширення базового типу (complexContent)

extension base = "...". Приклад

```
    <xs:element name="person">
      <xs:complexType>
      <xs:sequence>
      <xs:complexType name="client" minOccurs="1"  maxOccurs="unbounded">
        <xs:complexContent>
          <xs:extension base="Person">
            <xs:sequence>
            <xs:element name="passport" type="xs:string"/>
            </xs:sequence>
          </xs:extension>
          </xs:complexContent>
        </xs:complexType>
          <xs:complexType name="clerk">
               <xs:complexContent>
                  <xs:extension base="Person">
                    <xs:sequence>
                    <xs:element name="id" type="xs:integer"/>
                    </xs:sequence>
                </xs:extension>
                 </xs:complexContent>
              </xs:complexType>
```

```
            </xs:sequence>
          </xs:complexType>
    </xs:element>
    <xs:complexType name ="Person">
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="birthdate" type="xs:date"/>
          </xs:sequence>
</xs:complexType>
```

Розглянемо використання простих типів (simpleType).

```
<xs:simpleType name=”AgeType”>
   <xs:restriction base=”xs:integer”>
        <xs:minInlusive value=”5” />
         <xs:maxInlusive value=”99” />
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name=”birthday”>
   <xs:restriction base=”xs:date”>
        <xs:minInlusive value=”1950-01-01” />
         <xs:maxInlusive value=”2015-01-01” />
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name=”phone”>
   <xs:restriction base=”xs:string”>
        <xs:pattern value=”\d{3}-\d{2}-\d{2}”/>
   </xs:restriction>
```

Прості типи часто використовують обмеження - restriction (фільтри на дані) або регулярні вирази pattern value = "...". У нашому випадку телефон повинен мати формат 123-45-67. Можна зробити дефіс необов'язковим = "\d{3}-?\d{2}-?\d{2}"

Вельми корисним може бути використання перерахування

```
<xs:element name=”days”>
<xs:simpleType>
   <xs:restriction base=”xs:string”>
        <xs:enumeration value=”Неділя”/>
        <xs:enumeration value=”Понеділок”/>
        <xs:enumeration value=”Вівторок”/>
        <xs:enumeration value=”Середа”/>
        <xs:enumeration value=”Четвер”/>
        <xs:enumeration value=”П’ятниця”/>
        <xs:enumeration value=”Субота”/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

Всі можливості разширения та зміни типу у simpleType такі ж як і у complexType. Єдина відмінність - в complexType можна щось покласти, в простій тип -ні.

Контрольні питання.

1. Чи використовує HTML такий інструмент як DTD?
2. Чим відрізняется текст, що підлягає аналізу (парсінгу) від того, що не підлягає?
3. Чи відповідає парсінгу HTML-документ?
4. Що є валідацією?
5. Чи використовує HTML такий інструмент як XML-Schema?
6. Чи може комплексний тип даних включати в себе інші комплексні дані?
7. Яка мета об'єднаня елементів у группу?
8. Які функції фільтрів на дані?

Тема 3. XSLT (eXtendable Stylesheet Language)

XSLT як засіб перетворення XML, складається з

- eXtendable Stylesheet Language (розширювана мова опису стилів)
- XML Transformation (перетворення, засновані на XSL)

Для реалізації функції відображення браузером XML-файла існує процесінгова інструкція

<?xml-stylesheet type="text/css" href="file.css"?>
Дозволяє підключити каскадну таблицю стилів "file.css".

Наприклад, "book.css"
```
*{display: block;}
book{border-bottom: 1px dotted gray;}
title,author,price {display: inline; margin: 1ex;}
title{font-weight:bold;}
price{font-weight:bold;color:red;}
```

дозволяє оформити XML-документ.

* {display: block;} - все елементи є блоковими,

book {border-bottom: 1px dotted gray;} - елемент book потрібно підкреслити пунктирною лінією сірого кольору,

title, author, price {display: inline; margin: 1ex;} - елементи title, author, price є inline-елементами, тобто, рядковими, і між ними є невеликий відступ,

title {font-weight: bold;} - елемент title потрібно промальовувати жирним шрифтом,

price {font-weight: bold; color: red;} - а price не тільки жирним, а й червоним кольором.

Але використання XSLT в порівнянні з CSS - як небо і земля. XSLT незрівнянно могутніше і ефективніше. Перейдемо до розгляду XSLT.

Як і у CSS є "селектор", умова виконання правила при зустрічі зазначеного елемента, так і у XSLT, є умови спрацьовування при зустрічі зазначеного вузла XML.

<xsl:template match=”…условие…”>
Наприклад,

<xsl:template match=”book[price < 200]”>
Умова на те, що значення price дочірнього вузла від book менше 200.

Після того, як знайдено відповідний вузол, виконується перетворення, наступне за умовою вибору, тому й називається XML Transformation.

Тому один і той же файл XML за допомогою різних XSLT легко перетворити в інший XML, в HTML, RTF, FB2, просто в текст і т.д.

XSL таблиця - це XML документ з кореневим елементом stylesheet, всі елементи якого лежать в просторі імен

http://www.w3.org/1999/XSL/Transform

але XSL може використовувати будь простору імен.

```
<xsl:stylesheet version=”1.0”
    xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
                <xsl:template>
            …
       </xsl:template>
            …
</xsl:stylesheet>
```

Для виконання перетворення в браузері використовують процессинговую інструкцію

<?xml-stylesheet type = "text/xsl" href = "file.xsl"?>

Але можна провести перетворення в командному рядку

C:\>msxsl catalog.xml book.xsl -o result.txt

Шаблонні правила написані на спеціальній мові XPath, якому ми приділимо особливу увагу, так як це є ключовим елементом XML-перетворень.

Тобто, маємо шаблон (правило обробки XML-документа)

```
<xsl: stylesheet version = "1.0"
    xmlns: xsl = "http://www.w3.org/1999/XSL/Transform">
      <xsl: template match = "XPath вираз (умова співпадання)">
          Тіло шаблону
      </xsl: template>
</xsl: stylesheet>
```

І висновок значення

```
                <xsl: template match = "XPath вираз (умова співпадання)">
          <xsl: value-of select = "XPath вираз">
```

</xsl: template>

Наприклад, якщо в xsl файлі є запис

<xsl: template match = "book">

Hello!

</xsl: template>

І відповідний xml-файл має вузол book, то замість вмісту кожного такого вузла виведеться повідомлення "Hello!". Запис value-of виводить відібрані дані.

Питання: як відпрацює парсер в разі умови

<xsl: template match = "*">

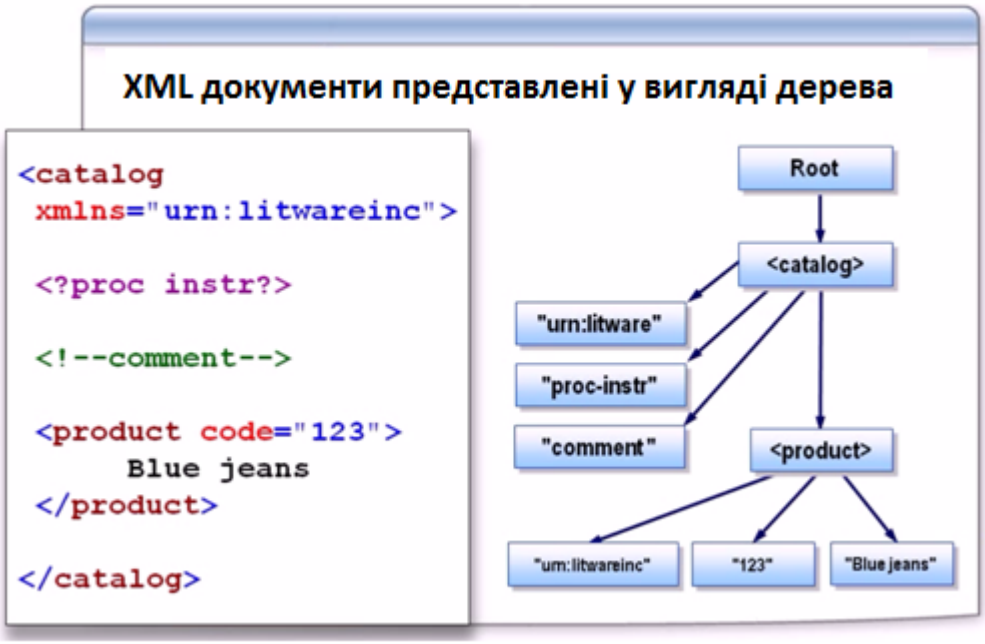
Відповідь - виведе всього один раз повідомлення "Hello!".

Причина в тому, що обробка одного кореневого тега - це обробка всього документа. Тобто все, що в кореневому тезі було замінено на "Hello!".

Якщо поставити <xsl: template match = "/"> то це буде стосуватися навіть не кореневого тега - а кореня, тобто те, що перед кореневим тегом, що дозволяє, гарантовано обробити весь документ цілком. Зауважимо, що, так як наявність необроблених вузлів може привести до краху, то визначення обробки кореня є корисним, всередину якого, як правило, ставлять інші процедури обробки.

XPath

Вирази XPath - це спосіб запису зазначення місця елемента в дереві XML. У XML є корінь (сам документ) і є кореневий елемент (document element).



Так як простір імен автоматично успадковується, в елемента product простір імен також є дочірнім елементом.

Вирази XPath нагадують навігацію по файловій системі, наприклад, шлях до файлу щодо поточного контенту, але на відміну від навігації по файловій системі, де ми обираємо один файл, вираз XPath вибирає всі вузли, які задовольняють умові

<xsl: template match = "job/title">

/ Абсолютний шлях, зазначений від кореня

<xsl: template match = "/ employee/job/title">

// Пошук елемента за будь-якої глибині

<xsl: template match = "employee // name">

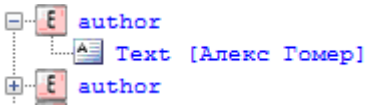
* -довільний елемент

<xsl: template match = "*">

Щоб проілюструвати роботу XPath, візьмемо Liquid XML Studio, виберемо з меню View-> XPath Query Builder. В випав вікні можна записувати XPath вирази, наприклад, якщо для документа catalog.xml набрати "/ catalog", то в нижньому вікні буде вказана структура цього елемента, а у верхньому - виділиться елемент catalog з усього документа, якщо/catalog/book, то ті ж дії будуть проведені для елементів book, для/catalog/book/author

```
<book>
        <author>Алекс Гомер</author>
        <title id='1'>XML и IE5</title>
        <pubyear>2000</pubyear>
        <price>200</price>
    </book>
    <book>
        <author>Алекс Хоумер</author>
        <title id='2'>Dimanic HTML</title>
        <pubyear>2004</pubyear>
        <price>120</price>
</book>
```

та



Більш того, використовуючи закладку Code Snippet, ми можемо отримати відповідний код на заданій мові програмування -C #, Java та ін.

Вибірка вузлів проводиться від поточного контексту, тому важливо вказати як (куди) проводиться пошук вузлів, що задовольняє умовам XPath - вліво-вправо, вгору або вниз по дереву документа. Для цієї мети існує таке поняття як осі вибірки.

Self вказує на поточний вузол. На наведеній DOM-моделі це вузол product.

Child-дочірні вузли price, discount. Code, начебто в дереві і дочірній елемент, але він атрибут.

Parent -батьки. У нашому випадку це catalog.

Attribute -це окремий тип вузлів, тому існує спеціальна вісь.

Descendant – ті, що стоять нижче (десант), тобто всі вузли, які по DOM-моделі нижче поточного.

Descendant-or-self - поточний і все спадний.

Ancestor -все висхідні вузли до кореня.

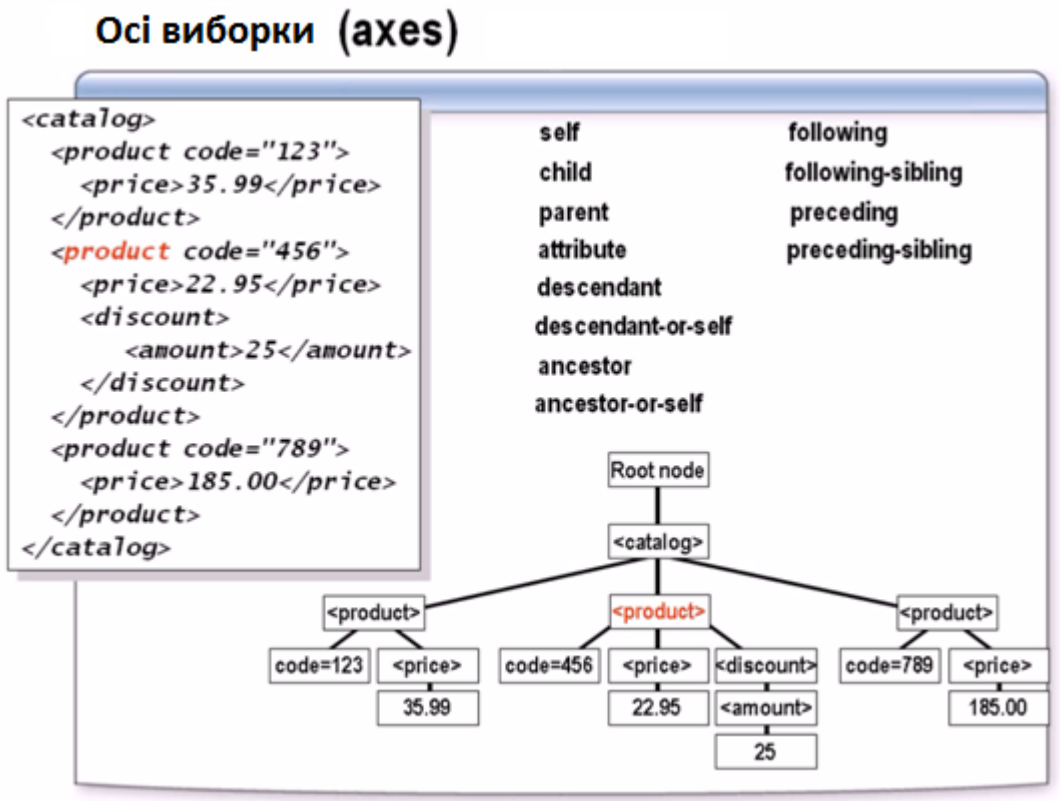
Ancestor-or-self - поточний і всі далі (нижче).

Following-все, наступні за поточним вузлом (з урахуванням того, що вузли обробляються в порядку проходження в XML-файлі, в DOM-моделі це зліва-направо).

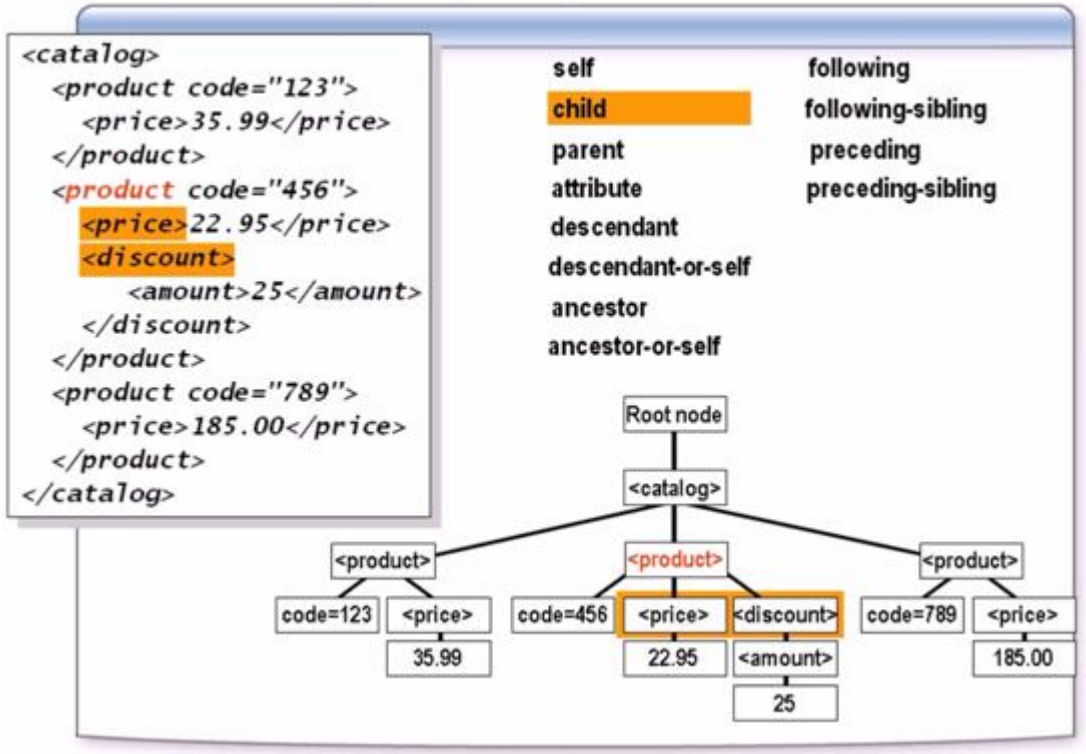
Following-sibling - наступні сусіди (тобто на одному рівні DOM-моделі).

Preceding –попередні.

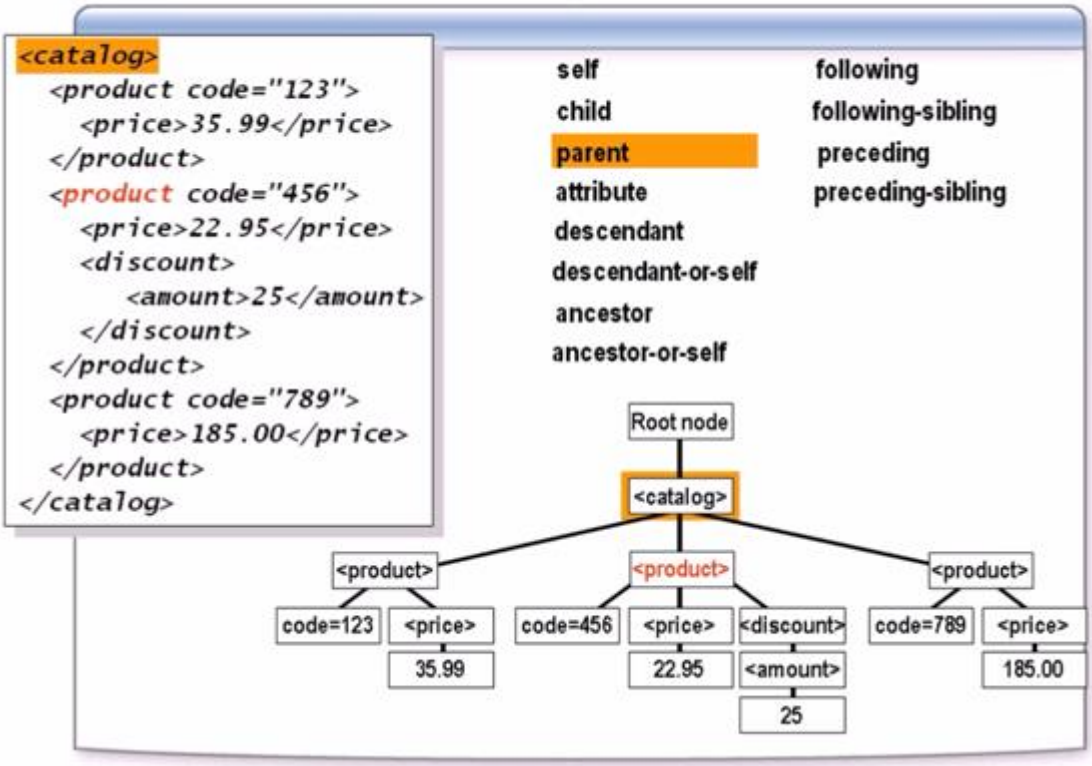
Preceding-sibling - попередні сусіди.



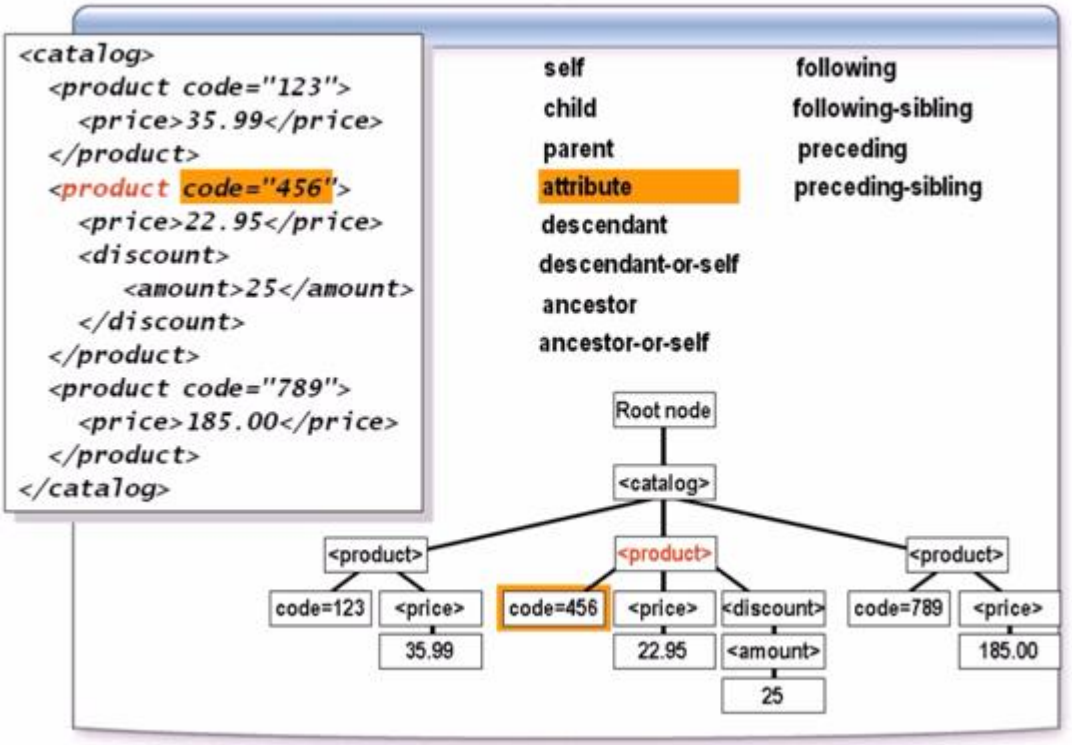
Осі виборки (axes)



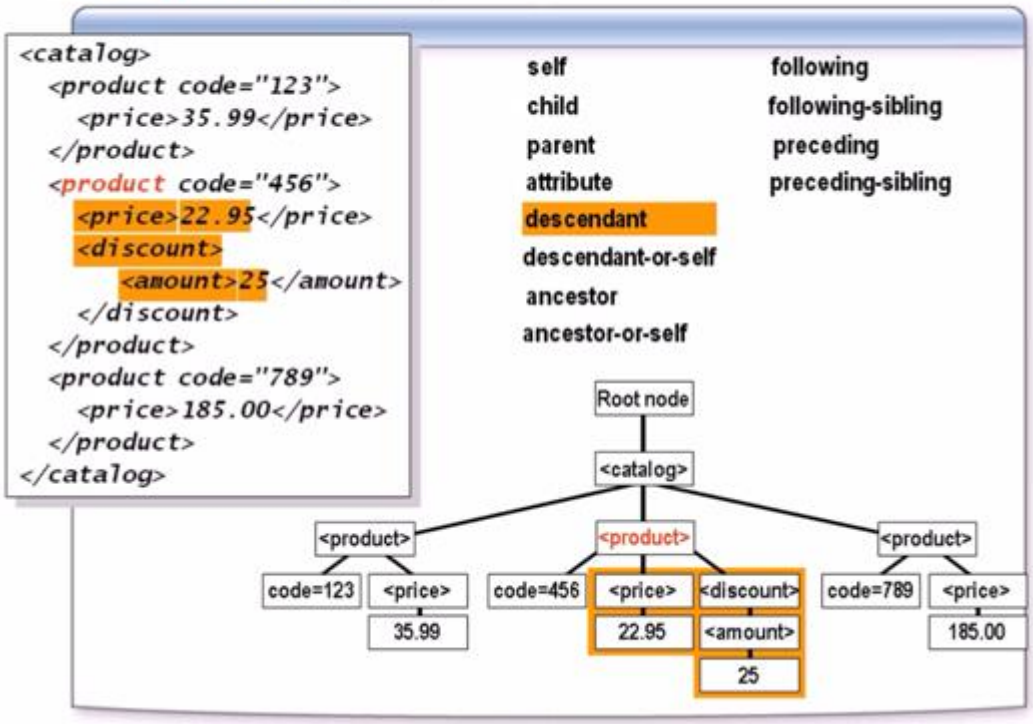
Осі виборки (axes)



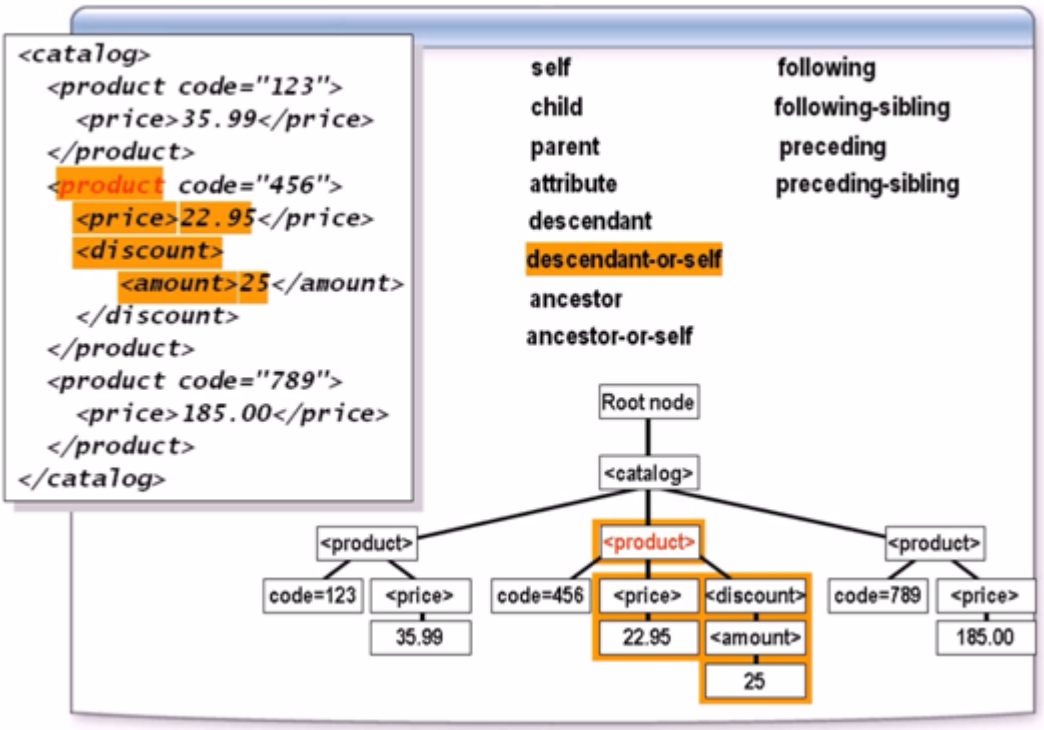
Осі виборки (axes)



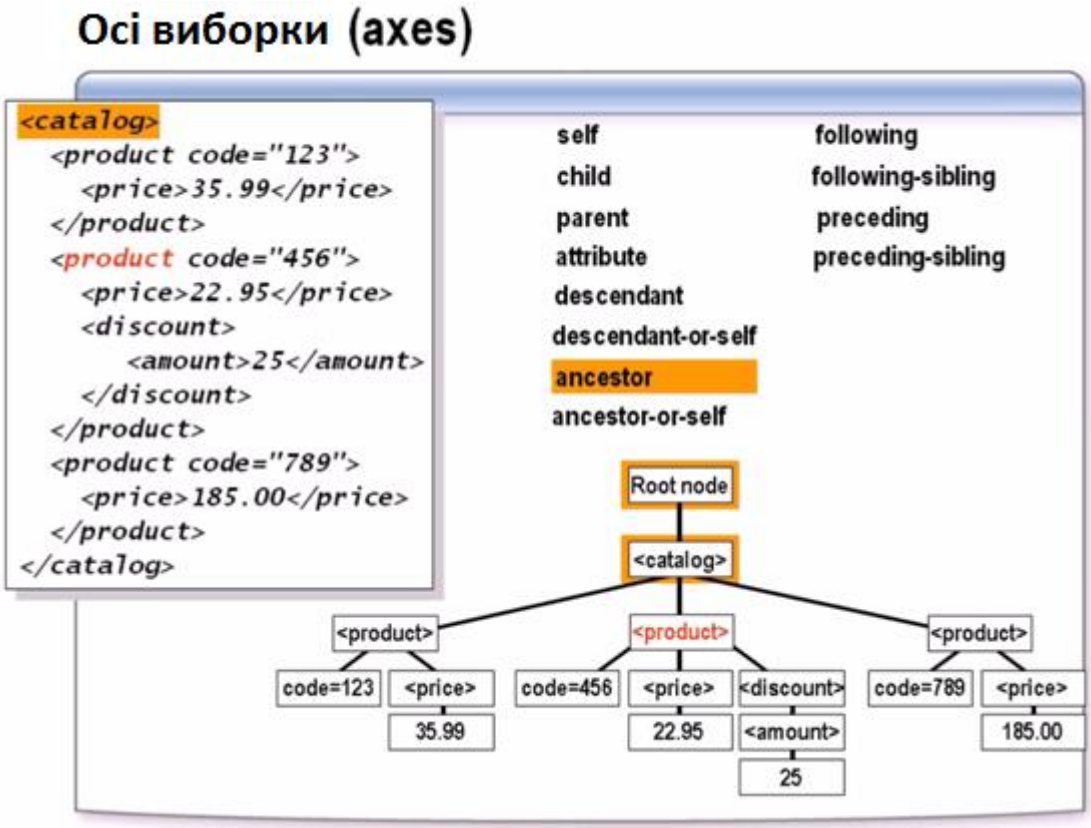
Осі виборки (axes)



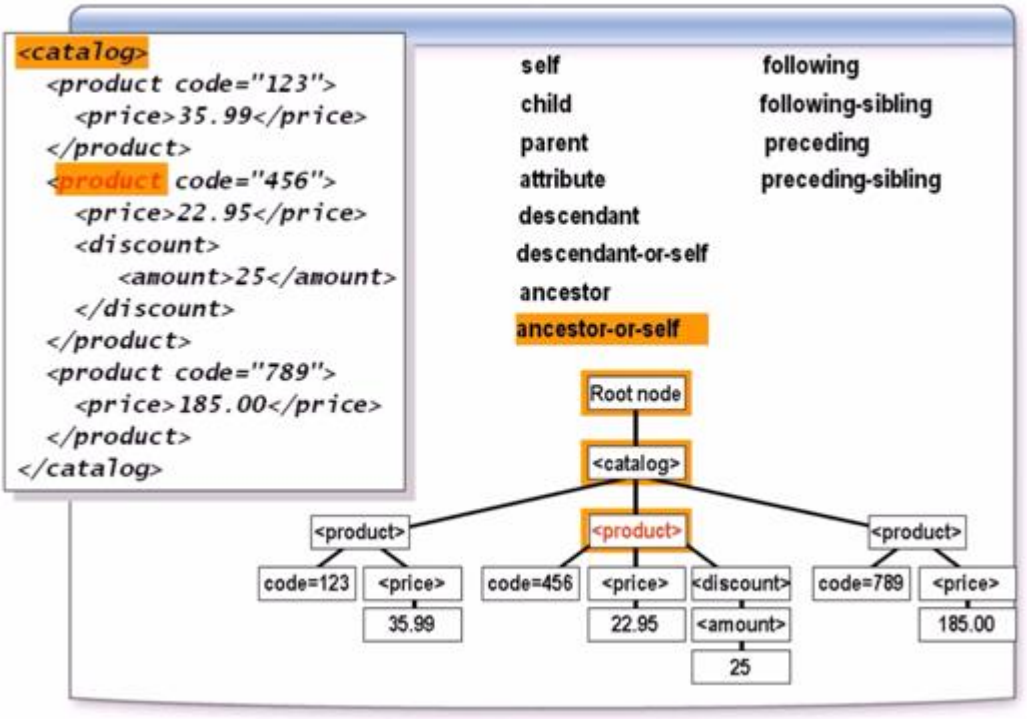
Осі виборки (axes)



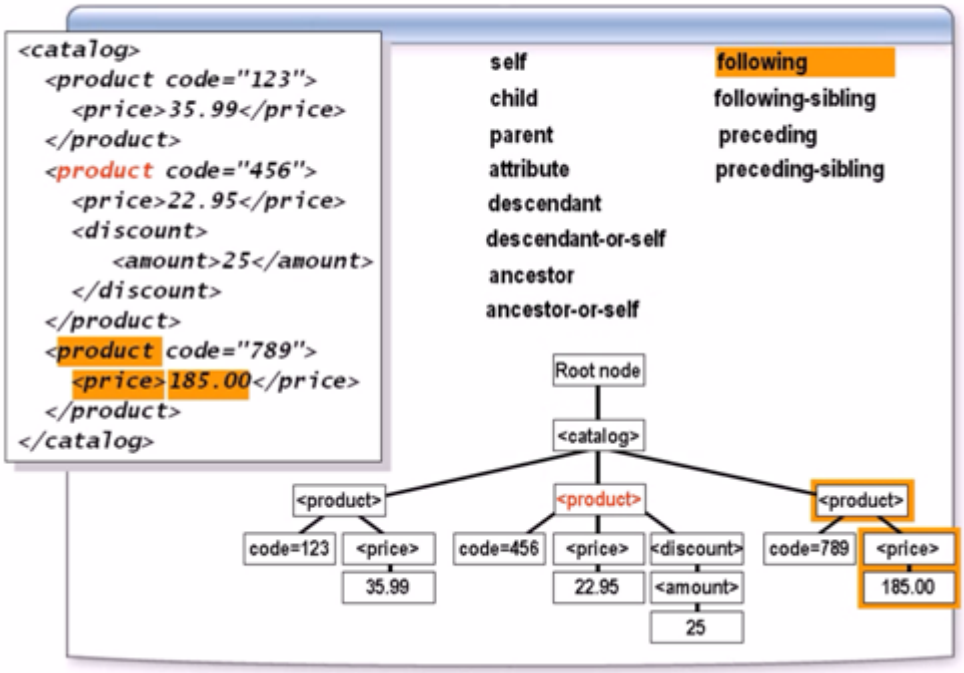
Осі виборки (axes)



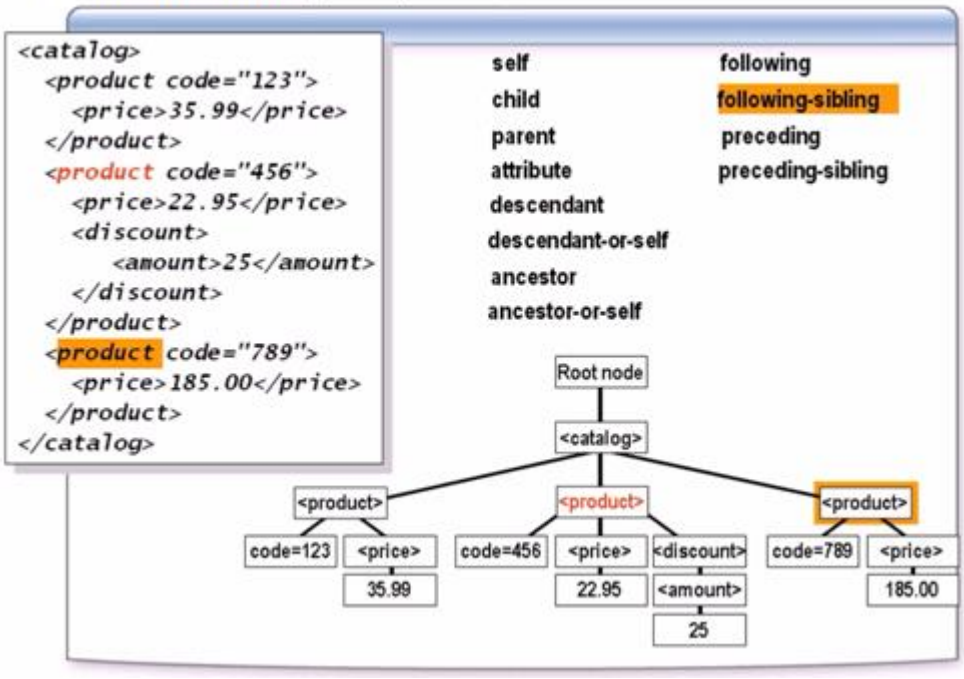
Осі виборки (axes)



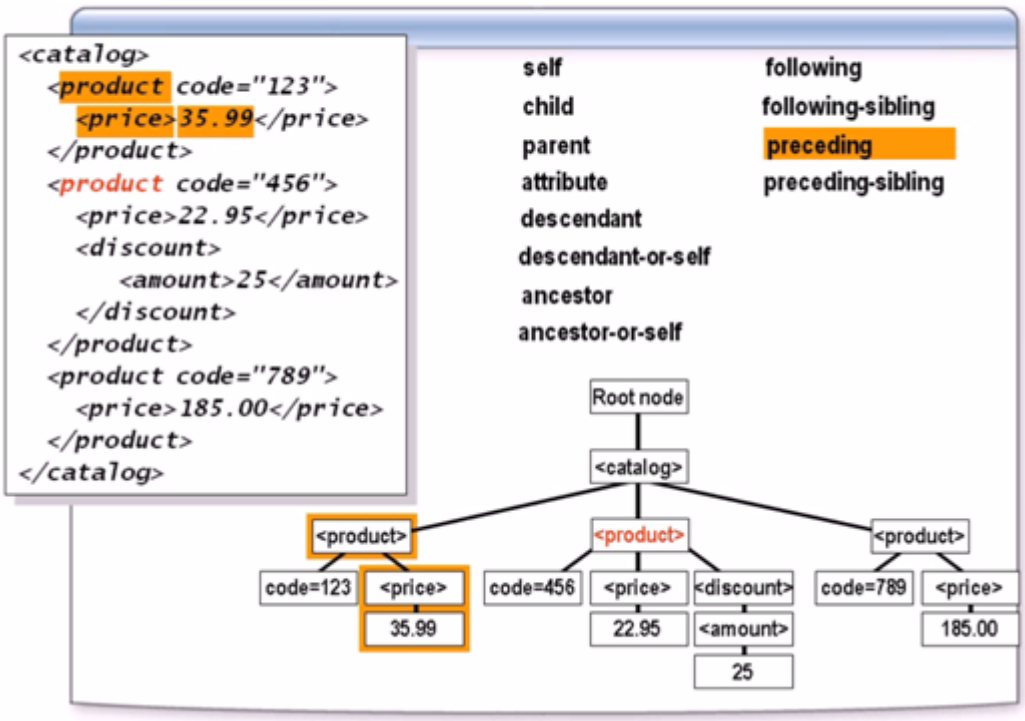
Осі виборки (axes)

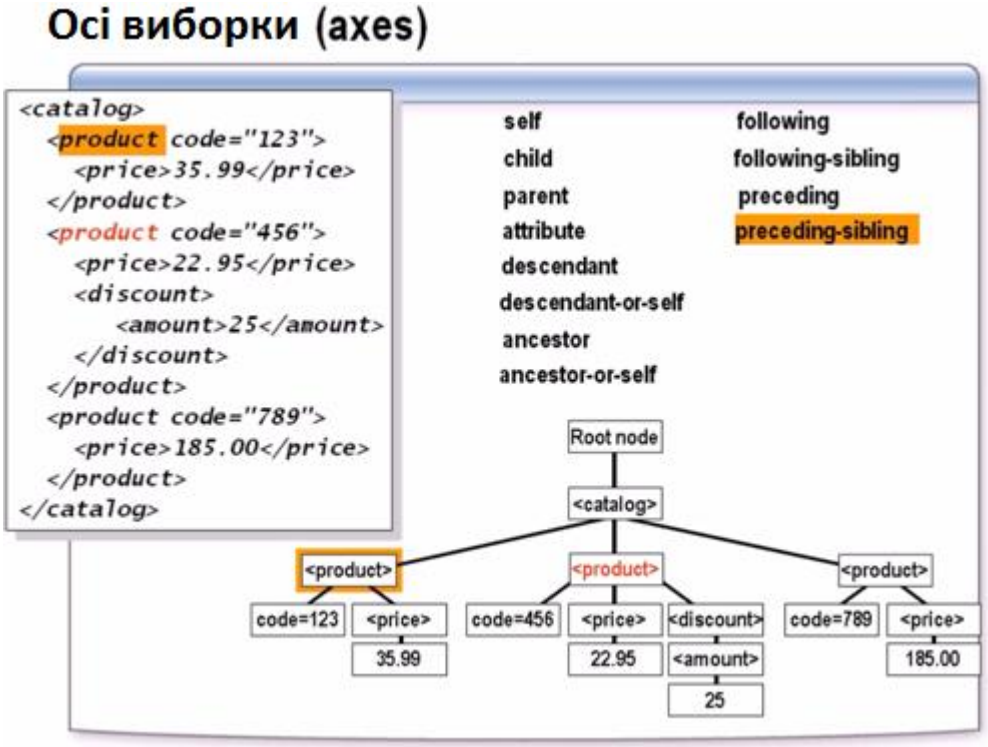


Осі виборки (axes)



Осі виборки (axes)





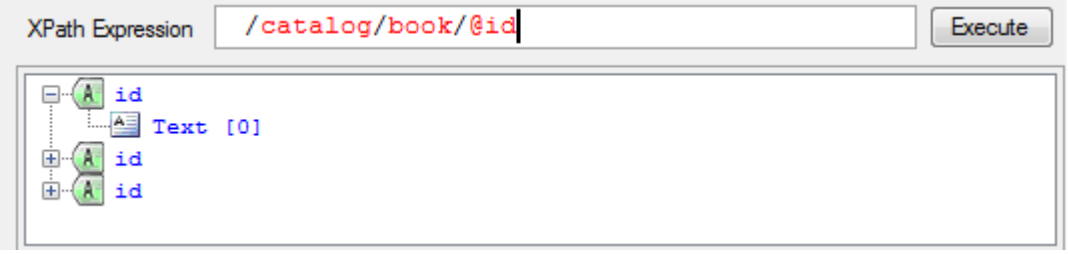
Скорочені записи вісей

Вісь	**Повний запис**	**Скорочений запис**
child	child::price	price
attribute	attribute::code	@code
self	self::node()	.
parent	parent::node()	..
descendant-or-self	/ descendant-or-self()/price	//price

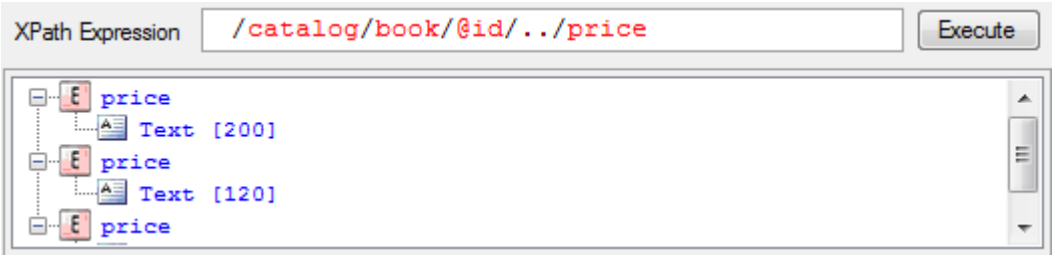
Тобто, запис "./author" (від поточного елемента дочірній елемент "author") і "author" - це одне і те ж

```
<?xml version =”1.0” encoding=”UTF-8” ?>
<xsl:stylesheet version=”1.0”
        xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
            <xsl:template match=”/catalog/book”>
                <xsl:value-of select=”./author” />
            </xsl:template>
</xsl:stylesheet>
```

Для пошуку по атрибутах, наприклад, елементу book, набираємо/catalog/book/@ id, і отримуємо вибірку елементів book, у яких є атрибут id.



Якщо візьмемо



то отримаємо вибірку елементів price у всіх елементах book, у яких є атрибут id. Вибрати всі вузли *, всі атрибути @ *, елемент простору імен aprefix: price, атрибути елемента простору імена prefix: code, всі вузли простору aprefix: *, всі атрибути простору aprefix: *.

ВАЖЛИВО: префікс простору імен -одна двокрапка «:», назва вісі вибірки - дві «::».

Для вибірки вузлів по типу використовують функції

text(), node(), processing-instruction(), comment().

Наберемо в полі XPath Query Builder програми Liquid XML Studio запис

/catalog/book/author/parent::node()

В результаті отримаємо всі батьківські вузли від author, тобто всі вузли book. Зауважимо, що функція node() повертає поточний вузол (поточний контекст обробки).

Якщо візьмемо/catalog/book/author/ancestor-or-self::node(), то виберемо автора, книгу, каталог і весь документ.

Інший приклад - отримання всіх коментарів

<?xml version = "1.0" encoding = "UTF-8"?>

<xsl: stylesheet version = "1.0"

xmlns: xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl: template match = "/">

<xsl: value-of select = "/ catalog/comment()" />

</xsl: template>

</xsl: stylesheet>

Потужність XPath полягає в можливості використання предикатів - фільтрів, які поміщаються в квадратні дужки

axis :: node-test [predicate ...]

Приклади фільтрації

product [last()] - положення вузла

product [@ id = "123"] - вміст вузла

product [id] - наявність вузла

Може бути кілька предикатів, обчислюваних зліва-направо

product [3] [@ id]

Так, /catalog/book[./price<200] дасть вибірку

```
5     <book id="0">
6         <author>Алекс Гомер</author>
7         <title id='1'>XML и IE5</title>
8         <pubyear>2000</pubyear>
9         <price>200</price>
10    </book>
11    <book id="1">
12        <author>Алекс Хоумер</author>
13        <title id='2'>Dimanic HTML</title>
14        <pubyear>2004</pubyear>
15        <price>120</price>
16    </book>
17    <book id="2">
18        <author>Алексей Валиков</author>
19        <title id='3'>Технология XSLT</title>
20        <pubyear>2006</pubyear>
21        <price>150</price>
22    </book>
```

Можна вказати номер вузла

/catalog/book [2]

Якщо вузол останній, то можна використовувати функцію

/catalog/book [last()]

Функція count() повертає кількість вузлів, тому можна використовувати запис

/catalog/book [count (/ catalog/book)]

ВАЖЛИВО: нумерація завжди починається з одиниці!

Функція position() повертає номер елемента, наприклад

/catalog/book [position() mod 2 = 0]

Повертає вузли з парними номерами.

При використанні декількох атрибутів завжди слід пам'ятати, що отримуються вони в порядку проходження в файлі, так запис

/catalog/book [2] [@ id]

вказує, що потрібно вибрати другий елемент book, якщо у нього є атрибут, а запис

/catalog/book [id] [2]

говорить про те, що потрібно вибрати всі елементи book, у яких є атрибути і з них вибрати другий.

Операції XPath

Логічні операції

A and B, A or B

Арифметичні операції

A + B, A-B, A * B, A div B, A mod B, -A

Операції порівняння

A = B, A! = B, A> B, A <B, A> = B, A <= B

Але немає операції заперечення! Але є функція not().

Важливо! Якщо ми пишемо вираз XPath в XML, то замість ">" потрібно писати "& gt;" а замість "<" потрібно "& lt;"

/catalog/book[./price & lt; 200].

Типи даних і функції XPath

- Булеві функції
- Числові функції
- Строкові функції
- Функції множин вузлів

Повний перелік наведено на http://www.w3schools.com/

Наведемо кілька найбільш популярних функцій XPath

Булеві функції:

- boolean (object) // false-порожній рядок (множина), решта -true
- not (boolean) // інвертує логічне значення
- true()
- false()

Числові функції

- number (object) // перетворення до числа (інакше NaN-Not at Number)
- sum (node-set)
- floor (number) // округлення до нижнього цілого
- ceiling (number) // округлення до верхнього цілого

- round (number) // округлення до цілого

Строкові функції

- string (object) // перетворення в рядок
- concat (string, string, string *) // * - к-во не обмежене
- start-with (string, string) // читаємо з заданого місця
- contains (string, string) // перевірка на вміст подстроки
- substring-before (string, string) // вирізати рядок до вказаної
- substring-after (string, string) // вирізати рядок після зазначеної
- sumstring (string, number, number *) // вирізати з вказаної позиції вказане число символів
- string-length (string?) // Довжина рядка
- normalize-space (string?) // Прибрати початкові і хвостові прогалини
- translate (string, string_1, string_2) //заміна одного набору символів string_1 на інший string_2

Функції множин вузлів

- last() // останній вузол
- position() // номер поточного вузла
- count (node-set) // число вузлів
- local-name (node-set?) // Локальне ім'я без простору імен
- namespace-uri (node-set?) // Ім'я простору імен вузла
- name (node-set?) // Ім'я (тега) вузла

Процесори XML дозволяють розширювати набір функцій користувача функціями (PHP, Java, JavaScript, C # та ін.).

Приклад. Нехай дано для обробки файл

```
<table>
    <row1 a=”1” b=”2”>
    <row2 a=”1” b=”2”>
    <row3 a=”1” b=”2”>
    <row4 a=”1” b=”2”>
    <row5 a=”1” b=”2”>
    <row6 a=”1” b=”2”>
    <row7 a=”1” b=”2”>
</table>
```

Відповідний запит буде мати вигляд

```
/table/* [starts-with (name(), 'row')]
```

Основні елементи XSLT

Насамперед зазначимо, що інструкція

```
<?xml-stylesheet type = "text/xsl" href = "file.xsl"?>
```

спрямована, насамперед браузерам для відображення документа. Процесори XML її просто не помічають.

XSL-файл являє собою таблицю (шаблон) перетворення XML-документа, зміст якої є у підміні вузла, що задовольняє XPath умові, заданих виразом (шаблоном).

Використання шаблонів

<xsl: template

name = "name" // ім'я шаблону

match = "pattern" // умова спрацьовування шаблону

mode = "mode" //

priority = "number">

<! - Content (<xsl: param> *, template)->

</xsl: template>

Якщо використовується значення match то шаблон спрацьовує автоматично при виконанні умови match. Якщо стоїть name, то спрацьовування проходить тільки при вказівці імені шаблону. Якщо процесор сортує вузли в порядку пріоритету, при умов іщо пріоритети не вказані - у порядку запису в документі. Якщо частина шаблонів задана з пріоритетом, а частина - без нього, то спочатку будуть поставлені ті, у яких присутнє значення пріоритету (задане цілим числом). Якщо значення пріоритету однакові, то вони упорядковуються в порядку проходження. У випадку, якщо при виконанні деяких умов потрібно застосовувати не всі шаблони, а тільки деякі з них, використовується mode (свого роду умова if). У цьому випадку одні й ті ж вузли можуть бути оброблені кілька разів. Приклад, застосуємо

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
       <xsl:output omit-xml-declaration="yes" indent="yes"/>
              <xsl:strip-space elements="*"/>
              <xsl:template match="num[position() mod 3 = 1]">
               <tr>
                     <xsl:apply-templates mode="copy" select= ". |
                     following-sibling::*[not(position() >2)]"/>
             </tr>
             </xsl:template>
        <xsl:template match="*" mode="copy">
              <td><xsl:value-of select="."/></td>
       </xsl:template>
     <xsl:template match="num"/>
</xsl:stylesheet>
```

Для файлу

```
      <nums>
       <num>01</num>
       <num>02</num>
       <num>03</num>
       <num>04</num>
       <num>05</num>
       <num>06</num>
```

```
    <num>07</num>
    <num>08</num>
    <num>09</num>
    <num>10</num>
  </nums>
```

Тоді результатом буде

```
<tr>
  <td>01</td>
  <td>02</td>
  <td>03</td>
</tr>
<tr>
  <td>04</td>
  <td>05</td>
  <td>06</td>
</tr>
<tr>
  <td>07</td>
  <td>08</td>
  <td>09</td>
</tr>
<tr>
  <td>10</td>
</tr>
```

У цьому перетворенні кожен елемент num починаючи з позиції не рівної 3*k + 1 (k-ціле) замінюється шаблоном з порожнім елементом, тобто не обробляється, при цьому

```
<xsl: apply-templates mode = "copy" select = ". |
following-sibling :: * [not (position()> 2)] "/>
```

означає: "Не застосовувати до обраних шаблонів вузлів, які звичайно застосовуються (в режимі без mode), але застосовувати шаблони, які знаходяться в режимі copy".

Елемент xsl: output дозволяє вказувати, яким хочуть бачити кінцеве дерево при виведенні. Якщо XSLT процесор виводить на друк кінцеве дерево, він повинен робити це так, як задано елементом xsl:output, проте сам висновок процесор виконувати не зобов'язаний, omit-xml-declaration вказує, чи повинен XSLT процесор виводити декларацію XML, значенням атрибута повинно бути yes або no, ну і following-sibling :: - повертає множину елементів на тому ж рівні, наступних за поточним.

Якщо ні одна умова не спрацьовує, то вузол вважається необробленим і копіюється на вихід. При цьому, якщо є необроблені вузли, це говорить про неякісну роботу з XML-документом. Це не помилка, це ознака некваліфікованого підходу. Тому, або в кінці поставити шаблон, який спрацьовує на все-що завгодно, другий, більш правильний, - відразу реалізувати обробку документа в цілому, наприклад,

```
<xsl:template match="/">
    <html>
```

```
                <body>
                    <xsl:apply-templates select=”/catalog/book” />
                </body>
            </html>
        </xsl:template>
        <xsl:template match=”book”>
            <div>
                <xsl:value-of select=”title” />
            </div>
        </xsl:template>
```

Говорить про те, що весь документ замінити на html-документ, після чого замість book зробимо блок, в якому відобразимо елемент title. При цьому вираз <xsl: apply-templates select ...> говорить «а тепер ще раз застосуємо, але інший шаблон». select дозволяє вказати що вибираємо, але якщо не писати, то це все одно, що select = "./*" -будь вкладене у поточного, а можна вказати конкретні умови, наприклад

<xsl: apply-templates select = "/ catalog/book [price & gt; 200] ">

У цьому випадку на другому прогоні документа будуть вибрані всі книги з ціною більше 200 грн.

Якщо є кілька умов, то спрацьовує те, яке першим виконується. Якщо ж візьмемо код

```
<<xsl:template match="book[price &gt; 200]" >

                <div style="color:red">

                    <xsl:value-of select="title" />

                </div>

            </xsl:template>

            <xsl:template match="book" priority="10" >

                <div>

                    <xsl:value-of select="title" />

                </div>

            </xsl:template>
```

Та, зважаючи на наявність пріоритету, виконається друга умова, а не те, що стоїть раніше, тобто, не перше.

Можна використовувати конструкцію <xsl: apply-imports /> в цьому випадку ми маємо можливість в документ з шаблонами підключити інші шаблони, які містяться в деякому файлі <xsl: import href = "file.xsl" />.

Іменовані шаблони

<xsl:template name=”templatename”>

<!-- Content: (<xsl:param>*,template)-- >

</xsl:template>

<xsl:call-template

name=”templatename”>

<!-- Content: xsl:with-param*-- >

</xsl:call-template>

Іменовані шаблони самі по собі не спрацьовують, тільки в тому випадку, якщо їх викликати за допомогою конструкції call-template.

За замовчуванням простір імен XSLT безіменний. Якщо потрібно обробити документ або конкретні вузли з використанням конкретного простору імен, то використовується імпортування простору імен. У XSLT додається простір імен з будь-яким префіксом

xmlns: a = http: //www.dstu.dp.ua/xml/name_space

і у самому шаблоні XPath пишеться з префіксом простору імен

<xsl: template match = "a: book">

Створення вузлів елементів

<xsl: element

name = "name"

namespace = "URI"

use-attribute-sets = "namelist">

<! - Content: template ->

</xsl:element>

Використання конструкції element дозволяє створювати вузол з обчислюваним ім'ям name = "name", яке може генеруватися XPath. Зауважимо, що запис name = "name", говорить про те, що очікується рядок, але якщо використовувати фігурні дужки {...}, то цей запис буде інтерпретуватися як вираз XPath, наприклад,

<xsl: element name = "{concat {'item', position()}}">

Даний запис створює вузол з ім'ям itemN, де N- номер позиції вузла.

Для створення вузлів-атрибутів використовується схожа конструкція

<xsl: attribute

```
   name = "name"
   namespace = "URI"
      <! - Content: template ->
</xsl: attribute>
```

Приклад, якщо у документа є параметр URL, то створення гіперпосилання на цей елемент:

```
<xsl: apply-templates select = "/ catalog/book [url]" />
```

Вибрати всі книги, у яких є вузол book з атрибутом url. При відображенні книги створимо тег гіперпосилання <a>, в ньому потрібно створити атрибут з ім'ям href, в який прописати вміст атрибута url із записом значення вузла title.

```
        <xsl:template match="/">
          <html>
                <body>
                      <xsl:apply-templates select="/catalog/book[@url]" />
                </body>
        </html>
                </xsl:template>
                <xsl:template match="book" >
                      <div>
                            <a>
                                  <xsl:attribute name="href">
                                        <xsl:value-of select="@url" />
                                  </xsl:attribute >
                                  <xsl:value-of select="title" />
                            </a>
                      </div>
                </xsl:template>
```

Але все це можна зробити коротше, використовуючи вирази XPath

<xsl:value-of select="title" />

Розглянемо код

<test>

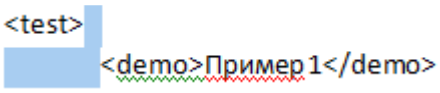
<demo> Приклад 1 </ demo>

<demo> Приклад 2 </ demo>

<demo> Приклад 3 </ demo>

</test>

Тут перший вузол - <test>, другий -текстовий вузол із символів



перенос рядка і знак табуляції, і т.д. Таким чином, у даному документі 7 вузлів!

І в кожного вузла <demo> ще один текстовий вузол «Приклад?».

Наскільки виправдане наявність текстового вузла без видимих символів. Для цієї мети в парсером існує властивість PreserveWhitespace. Якщо встановлено значення false, то такий текст як вузли не вважається. Проблема в тому, що у кожного парсеру (Microsoft, PHP, Mozilla та ін.) Значення встановлене за замовчуванням - різне.

Створення текстових вузлів

<xsl: text

disable-output-escaping = "yes | no">

<! - Content: #PCDATA ->

</xsl: text>

Приклад створення текстового вузла

<xsl: text> </xsl: text>

<xsl: value-of select = "price" />

<xsl: text> грн. </xsl: text>

Якщо потрібно вставити текст, який не підлягатиме парсингу, можна це зробити, наприклад, так

<xsl: text> <! [CDATA [& nbsp;]]> </xsl: text>

Але при цьому символ & перетвориться в amp; і символ прочитається неправильно. Для того, щоб все відобразилося правильно, використовується конструкція disable-output-escaping

Якщо її значення встановлено в "yes", то перетворення в безпечний код проводитися не буде.

У випадку, якщо необхідно ввести коментар, можна використовувати

<xsl: comment> Коментар </xsl: comment>

Копіювання вузлів

<xsl: copy

use-attribute-sets = "name-list">

<! - Content: template ->

</xsl: copy>

служить для копіювання поточного вузла і більш загальна конструкція, яка використовує XPath

<xsl: copy-of select = "expression" />

Наприклад, якщо потрібно з документа вибрати всі гіперпосилання, то це можна зробити наступним чином

<xsl: output method = "xml" />

<! - На виході формуємо xml-документ ->

<xsl: template match = "/">

<! - Обробити весь xml-документ ->

<result>

<xsl: apply-templates select = "// xhtml: a" />

<! - Рекурсивно від поточного контенту (по суті від кореня) потрібно

пройти по всьому документу і вибрати всі гіперпосилання простору імен xhtml ->

</result>

</xsl: template>

<xsl: template match = "xhtml: a">

<xsl: copy-of select = "." />

<! - Вибрані елементи "xhtml: a", тобто гіперпосилання, скопіруй ->

</xsl: template>

У XSLT крім можливості маніпуляції з вузлами, є керуючі конструкції.

Умовний перехід

<xsl: if test = "expression">

<! - Content: template ->

</xsl: if>

Приклад. Потрібно вивести ті товари, які є в наявності, тобто у яких присутня атрибут onstore = "yes"

<xsl: template match = "/">

<xsl: apply-templates select = "/ store/item" />

<! - Виберемо всі вузли item ->

</xsl: template>

<xsl: template match = "item">

<! - Піти по всім вибраним вузлам item ->

<xsl: value-of select = "@ name" />

<! - Надрукувати назву товару ->

<xsl: if test = "@ onstore = 'yes'">

В наявності

</xsl: if>

</xsl: template>

Відповідний xml-документ має вигляд

<store>

<item name = "Комп'ютер" color = "Чорний" onstore = "yes" />

<item name = "Монітор" color = "Срібний" />

<item name = "Монітор" color = "Червоний" onstore = "yes" />

</store>

Оператор вибору

```
<xsl:choose>
    <xsl:when test=”Boolean-expression”>
        <!—Content:template-->
    </xsl:when>
    <xsl:when test=”Boolean-expression”>
        <!—Content:template-->
    </xsl:when>
    <xsl:otherwise>
        <!—Content:template-->
    </xsl:otherwise>
</xsl:choose>
```

Для прикладу візьмемо попередній xml-документ і виведемо назве товару кольором самого товару

```
<xsl:template match="/">
        <xsl:apply-templates select="/store/item" />
<!-- Выберем все узлы  item -->
  </xsl:template>
  <xsl:template match="item">
        <xsl:choose>
            <xsl:when test="@color='Червоний'">
                <xsl:attribute name="style">
                    background-color="red";
                </xsl:attribute>
            </xsl:when>
            <xsl:when test="@color='Срібляний'">
                <xsl:attribute name="style">
                    background-color="gray";
                </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name="style">
                    background-color="white";
                </xsl:attribute>
            </xsl:otherwise>
            </xsl:choose>
            <xsl:value-of select="@name" />
</xsl:template>
```

Циклічна обробка

```
<xsl:for-each
    select=”expression”>
    <!—Content: (xsl:sort*,template)-->
</xsl:for-each>
```

Можна вирішити ту ж завдання, що і раніше, але з використанням циклу (без apply-templates)

<xsl: for-each select = "/ store/item">

<! - Для всіх обраних вузлів item ->

<xsl: choose>

<xsl: when test = "@ color = 'Червоний'">

<xsl: attribute name = "style">

background-color = "red";

</xsl: attribute>

</xsl: when>

<xsl: when test = "@ color = 'Срібний'">

<xsl: attribute name = "style">

background-color = "gray";

</xsl: attribute>

</xsl: when>

<xsl: otherwise>

<xsl: attribute name = "style">

background-color = "white";

</xsl: attribute>

</xsl: otherwise>

</xsl: choose>

<xsl: value-of select = "@ name" />

</xsl: for-each>

Рекурсія

<xsl: template match = "myElement">

. . .

<xsl: apply-templates select = "myElement" />

. . .

</xsl: template>

Розглянемо приклад. Нехай у нас є xml-документ, що містить меню з різним ступенем вкладення.

```
<menu>
    <menuItem title="Пошукові системи">
        <menuItem title="Гугл" link="http://www.google.com" />
        <menuItem title="Бінг" link="http://www.bing.com" />
        <menuItem title="Мета" link="http://www.meta.ua" />
    </menuItem>
    <menuItem title="Інформаційні ресурси">
        <menuItem title="Українські ресурси">
        <menuItem title="УкрНет" link="http://www.ukr.net" />
            <menuItem title="ЦензорНет"
            link="http://www.censor.net" />
            <menuItem title="Інформаційний спротив"
            link="http://www.sprotiv.info" />
    </menuItem>
        <menuItem title="Закордонні ресурсі">
        <menuItem title="BBC" link="http://www.bbc.co.uk" />
        <menuItem title="Радіо Свобода"
```

```
            link="http://www.radiosvoboda.org" />
            <menuItem title="Голос Америки"
            link="http://www.ukrainian.voanews.com" />
        </menuItem>
        </menuItem>
</menu>
```

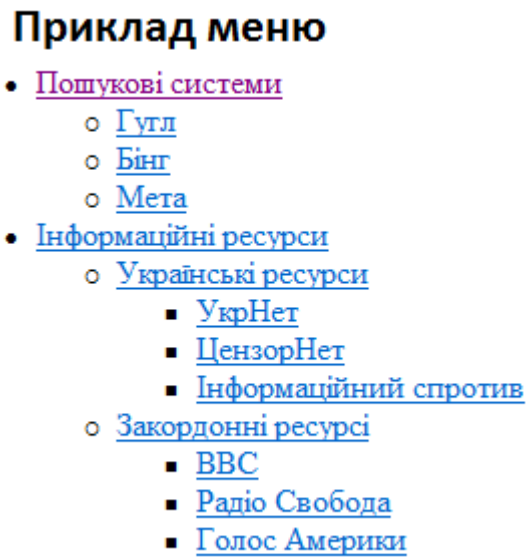
Треба сформировати html-документ. Для цієї мети будемо використовувати xslt

```
<xsl:output method="html" />
    <xsl:template match="/">
        <html>
        <head>
            <title> Приклад меню</title>
        </head>
            <body>
            <h1>Приклад меню</h1>
            <ul>
            <xsl:apply-templates select="/menu/menuItem" />
            <!-- Вибрати шаблони на першому рівні вкладення
```

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="rec.xsl"?>
<menu>
    <menuItem title="Пошукові системи">
    <menuItem title="Інформаційні ресурси">
</menu>
```

```
            -->
            </ul>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="menuItem">
            <!-- Для всіх обраних вузлів menuItem -->
        <li>
            <a href="{@link}">
                <xsl:value-of select="@title" />
            </a>
            <ul>
                <xsl:apply-templates select="./menuItem"/>
      <!--  виконується для поточного контенту або просто
            <xsl:apply-templates select="menuItem"/> -- >
                <!--Рекурсія! -->
            </ul>
        </li>
    </xsl:template>
```

В результаті отримуємо



Сортування

<xsl: sort

select = "expression" // Вираз XPath

lang = "language-code" // використована мова

data-type = "text | number | qname" // qname-спеціальні розширення, наприклад, сортування за датою тощо. (залежить від xslt-процесора. За замовчуванням - текст.

order = "ascending | descending" // зростання-спадання

case-order = "upper-first | lower-first" /> // від верхнього або нижнього регістра.

Приклад. Нехай необхідно відсортувати книги за ціною або за автором.

<xsl: template match = "/">

<xsl: apply-templates select = "/ catalog/book">

<! - Вибрати всі книги ->

<xsl: sort select = "price" data-type = "number" />

<! - Фільтр по атрибуту ./price як за кількістю (інакше як рядок). А можна відсортувати і по авторам ->

<xsl: sort select = "author" />

</xsl: apply-templates>

</xsl: template>

<xsl: template match = "book">

<! - Тепер по кожній книзі ->

<xsl: for-each select = "*">

<! - По всіх вкладених елементам ->

<xsl: value-of select = "name()" />

<! - Вивести назву тега поточного елемента ->

<xsl: text>: </xsl: text>

<! - Створити текстовий вузол із значенням «:» ->

<xsl: value-of select = "." />

<! - Вивести значення поточного контексту ->

</xsl: for-each>

</xsl: template>

Ключі та вибірка вузлів по ключу

Визначення ключа

<xsl: key

name = "name" match = "pattern" use = "expression" />

Вибірка по ключу

key ('keyName', 'value')

Розглянемо завдання: вибрати всі книги з ціною більше 200 грн.

<xsl: value-of select = "count (/ catalog/book [price & gt; 200])" />

<! - Пройти по всій колекції і вибрати ті, у яких ціна більше 200 ->

<xsl: apply-templates select = "/ catalog/book [price & gt; 200]" />

<! - Та ж вибірка вузлів, що і раніше !!! ->

</xsl: template>

<xsl: template match = "book">

<xsl: value-of select = "name()" />

<xsl: text>: </xsl: text>

<xsl: value-of select = "." />

</xsl: template>

У даному коді одна і теж дія проводиться двічі, що для великих документів може істотно уповільнити обробку документа. У базах даних для прискорення роботи існує поняття індексу (заздалегідь проведена обробка даних), завдяки чому дані можна індексувати, що істотно прискорює роботу програми. У XSLT для цієї мети

існує вибірка по ключу. Вибірка по ключу дозволяє заздалегідь зробити по заданій умові якусь операцію з вузлами.

Визначення ключа проводиться в самій таблиці стилів, але не в шаблоні. Ключів може бути кілька, кожен з яких повинен мати унікальне ім'я. Параметр match вказує які вузли потрібно повертати, use вказує умову щодо контексту match, за яким, власне кажучи, і проводиться індексація.

<xsl:key name="bookAuthor" match = "/catalog/book" use="author"/>

А потім, замість виклику вираження XPath, використовуємо ключ

<xsl: apply-templates

select = "key ('bookAuthor', 'Алекс Гомер')" />

Виклик ключа має два параметри, перший - його ім'я, другий вказує за яким критерієм потрібно повернути.

Використання ключа вимагає виконання строгої рівності виконання умови, не можна вимагати виконання нерівності, тобто якщо ключ

<xsl: key name = "mykey" match = "/ catalog/book" use = "price" />

то ми можемо вимагати виконання умови, що ціна дорівнює, наприклад, 200 грн.

select = "key ('mykey', 200)"

Ключі можуть бути складовими, коли потрібно вимагати виконання кількох умов.

Складові ключі.

<xsl: key name = "ixBookTitleAuthor"

match = "/ catalog/book"

use = "concat (title, author)" /> // Ось вони, складові умови

. . .

<H2> Книги XML, написані Гомером Сімпсоном </ h2>

<xsl: apply-templates

select = "key ('ixBookTitleAuthor',

concat ('XML', 'Гомер Сімпсон')) "/>

і ще раз. Використання ключів виправдано тоді, коли одна і та ж віборка використовується багаторазово.

Управління виводом документа

<xsl: output method = "xml | html | text | name"

version = "string"

encoding = "string"

omit-xml-declaration = "yes | no"

standalone = "yes | no"

doctype-public = "string"

doctype-system = "string"

cdata-section-element = "namelist"

indent = "yes | no"

media-type = "string" />

Оператор управління виводу глобальний, тому повинен бути на рівні таблиці перетворень, а не в шаблонах.

Головний атрибут - method, який вказує вигляд вихідного документа і має три стандартних значення xml | html | text (тільки текстові вузли без тегів) і додатковий - name, що залежить від конкретного xml-процесора.

version використовується тільки для виведення xml. За замовчуванням використовується 1.0. encoding вказує кодування вихідного потоку (для різних xml-процесорів може працювати некоректно, краще використовувати UTF-8, це гарантовано).

omit-xml-declaration придушити вивід xml-декларації.

standalone -самодостатність документу (застаріле).

doctype-public і doctype-system дозволяють зробити посилання на DTD.

cdata-section-element = список заборонених символів. При обробці вони потрапляють в <! [CDATA ["Список заборонених символів"]]>

indent вказує чи робити відступи у вихідному потоці, або все записувати в один рядок.

media-type вказує для браузера якого типу дані формуються.

Таким чином, типові випадки можна описати так

```
<xsl:output method=”text”
        encoding=”utf-8” />
<xsl:output method=”html”
   doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
  doctype-system=”http://www.w3.org/TR/html4/strict.dtd”
        cdata-section-element=”namelist”
        indent=”yes” />
<xsl:output method=”xml”
        doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
        doctype-system=”http://www.w3.org/1999/xhtm”
        indent=”yes”/>
```

Це висновок xhtml, але його в методах немає, хоча це xml.

Використання декількох вхідних документів (вихідний завжди один)

Функція document ('url')

<xsl: copy-of select = "document ('my.html')/html/body/h1" />

Наприклад,

```
<xsl:template match="message">
<xsl:value-of select="document('forum_user.xml')
/users/user[@login=current()/@author]" />
```

У функції document всього один параметр - URL документа, шлях може бути як абсолютний (http: /...), так і відносний. В результаті завантажується цей документ і створюється ще одна DOM-модель, яку можна використовувати. Далі йде XPath вираз по DOM-моделі нового документа. У даному прикладі - в завантаженому документі є/users/user але порівняти потрібно по атрибуту login, значення котрого має співпадати зі значенням атрибута author нової DOM-моделі, ніби як повинна виконуватися умова

[login =./@ Author]

але, ./ позначає текучий контекст, а текучий контекст визначається співвідношенням

<xsl: template match = "message">

тобто, message, а предикат [...] перебираючи всі вузли, поточним вже вважає user.

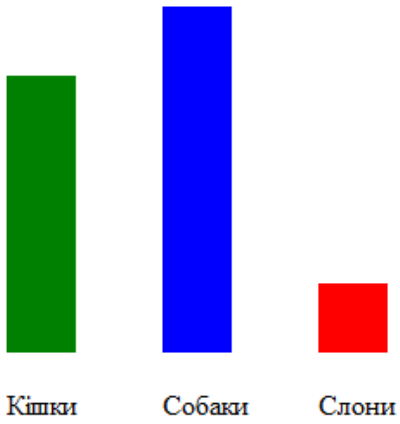
Для цієї мети служить функція current() яка повертає поточний контент самого шаблону, в нашому випадку це message. Іншими словами, цю умову можна записати

./@login=current()/@author

Тобто, для кожного author шукаємо значення login яке співпадає з ним.

Якщо параметр функції - пустий рядок document (") то в цьому випадку завантажується сам xslt-документ, за яким будується DOM-модель. Зауважимо, що елементи для побудови DOM-дерева повинні лежати на рівні таблиці стилів, в просторі імен відмінного від використовуваного за замовчуванням. Використання перезавантаження самого xslt-документа виправдано в тому випадку, якщо на вимогу безпеки системи існує заборона на завантаження зовнішніх файлів, в цьому випадку перезавантаження стильового файлу в якості контейнера для власних даних може вирішити проблему.

Зауважимо, що виведення даних не обов'язково може бути текстом, це може бути графік, діаграма тощо (див., Приклад, animal, де діаграми малюються з використанням блоків div.)



де вираження XPath включені прямо в стильові записи

Додаткові функції

При завантаженні вихідного документа xml-процесором при побудові DOM-дерева кожному вузлу встановлюється своє значення id, що є внутрішньою службовою функцією процесора. При цьому наступне перетворення породить зовсім інші значення цих id. Для виведення цих значень існує функція generate-id.

<xsl: value-of select = "generate-id()" />

Якщо функція запускається без параметрів, вона повертає ідентифікатор поточного контексту, але у неї може бути і параметр, який вказує на конкретний вузол, так generate-id (..) повертає ідентифікатор батьківського вузла, якщо ж вказати як параметр набір вузлів, то вона повертає ідентифікатор першого вузла з цього набору.

generate-id (/ catalog/book) поверне ідентифікатор першої книги.

Змінні і параметри

Одним з методів прискорення роботи XSLT, при обробці повторюваних даних, є використання змінних, в яких зберігаються значення проміжних обчислень.

```
<xsl:variable
        name=”name”
        select=”expression”>
        <!—Content:template -->
</xsl:variable>
```

У XSLT **не існує** оператора присвоєння значень, тому не можна створити змінну і потім присвоїти їй значення, можна створити змінну з відомим значенням, по суті, це створення константи.

Область видимості змінної обмежена елементом, в якому створена ця змінна. Змінна може приймати значення будь-якого вузла, набору вузлів, значення (рядок, число), фрагмент XML-коду і т.д.

Значенням змінної може бути значення самого елемента, наприклад,

<xsl: variable name = "hello"> Hello World! </xsl: variable>,

тобто в саму змінну записуємо її значення.

Можна визначити вузол

```
<xsl:variable name="AnimalCollection">
        <animals>
          <animal name="Собака" />
          <animal name="Кішка" />
        </animals>
    </xsl:variable>
```

Змінну можна використовувати в будь-якому XPath вираженні, випереджаючи ім'я змінної знаком долара «$», наприклад,

<xsl:value-of select="$hello" />

дозволяє вивести рядок «Hello World!».

Звернемося тепер до більш складної змінної "animals" і виведемо перший елемент

<xsl: value-of select = "$ AnimalCollection/animals/animal [1]/@ name" />

Інший спосіб визначення змінної полягає у визначенні XPath виразу

<xsl:variable name="myBook" select="/catalog/book[contains(title, 'XML')]"/>

тобто змінна створена порожньою і її значення заповнюється результатом виконання XPath виразу. У даному виразі з catalog вибираються всі елементи book у яких в назві title є підрядок 'XML', тобто, вибираються є всі елементи для яких функція contains повертає true. При цьому дана дія проводиться під час ініціалізації змінної, а не при її виклику.

Виклик здійснюється, наприклад, наступним чином

<xsl: text> Середня ціна </xsl: text>

<xsl: value-of select = "sum($myBook/price) div count($myBook)" />

Приклад використання змінних з заданим деревом вузлів

```
<xsl:variable name="header">
    <tr>
        <th>Element</th>
        <th>Description</th>
    </tr>
</xsl:variable>

<table>
    <xsl:copy-of select=”$header” />
    <xsl:for-each select=”reference/record” />
</table>
```

Параметри

```
<xsl:param
        name=”name”
        select=”expression”>
        <!—Content template -->
<xsl:param>

<xsl:call-template name=”show_title”>
        <xsl:with-param name=”title”/ >
</xsl:call-template>
```

Як зазначалося раніше, шаблони бувають звичайні (спрацьовує автоматично) та іменовані (викликаються за запитом імені шаблону). Параметри - це змінні, які можуть бути задані в момент виклику шаблону (аналогічно параметрам функції).

При цьому значення параметра можуть бути порожніми

<xsl: template name = "textBlock">

<xsl: param name = "text" />

або ж заповненими

<xsl: param name = "background" select = "'# fff'" />

в цьому випадку - в подвійних лапках стоїть вираз XPath, а в одинарних -строка, при цьому можна записати дещо в іншому вигляді

<xsl: param name = "background"> #fff </xsl: param>

Параметри викликаються як звичайні елементи

<xsl: value-of select = "$ title" />

А шаблон запускається з параметрами

<xsl: call-template name = "textBlock">

<xsl: with-param name = "title" select = "title" />

<xsl: with-param name = "text" select = "concat (author, ", price)" />

</xsl: call-template>

У разі необхідності вибірки неповторяющихся значень можна використовувати наступну конструкцію.

Вибірка унікальних значень

- групування- вибірка унікальних елементів з набору
- Завдання групування знайти множину елементів з однаковим значенням і взяти перший елемент множини
- Перший елемент, це елемент що не має попереднього (сусід на одному рівні відсутній).

preceding-subling :: item [... умова ...] == null

Інший підхід до вирішення цього завдання - угруповання Мюнха (Stive Muench з ОРАКЛ)

- Функція generate-id повертає ID першого елемента множини
- За допомогою ключів можна вибрати множину вузлів за їх властивостями.

Розглянемо задачу. З файлу, що містить інформацію про книжки, вибрати всіх авторів без повторення або, іншими словами, провести групування книжок по авторам.

Спочатку створимо ключ

<xsl: key name = "idxAuthor" match = "/catalog/book/author" use ="." />

Повертає елемент author за значеннями самого author. У цьому випадку функція

key ('idxAuthor', 'Алекс Гомер')

поверне набір вузлів (книги) з автором «Алекс Гомер». Вузловий для даної задачі є конструкція

<xsl: variable name = "authors"

select = "/ catalog/book/author [generate-id (.) =

generate-id (key ('idxAuthor',.))] "/>

У цьому випадку/catalog/book/author [generate-id (.)] проходить по всіх вузлах авторів, повертаючи унікальні ID кожного вузла? А XPath умова

generate-id (.) = generate-id (key ('idxAuthor' ,.)

ідентифікатор поточного вузла дорівнює ідентифікатору, поверненого функцією key для всіх поточних значень author, а generate-id повертає ідентифікатор тільки першого елемента набору. Тобто ідентифікатори (в силу унікальності) збігаються тільки в тому випадку, коли це перший автор у списку, але навіть при збігу найменування авторів, якщо це не перший автор в наборі, їх ідентифікатори не збігатимуться.

Такий прийом корисний при виконанні угруповання за автором. У цьому випадку спочатку визначаємо унікального автора, потім вибираємо всі книги, що належать даному автору

<xsl: for-each select = "/ catalog/book [author = $ currAuthor]/title">

<xsl: value-of select = "." />

</xsl: for-each>

Зауважимо, що в цьому випадку ми ще раз пробігаємо по всьому документу, вибираючи елемент author, хоча це було зроблено раніше при формуванні ключа. Таким чином, можна це ж дію прискорити, використовуючи наявний ключ

<xsl: for-each select = "key ('idxAuthor', $ currAuthor) /../ title">

<xsl: value-of select = "." />

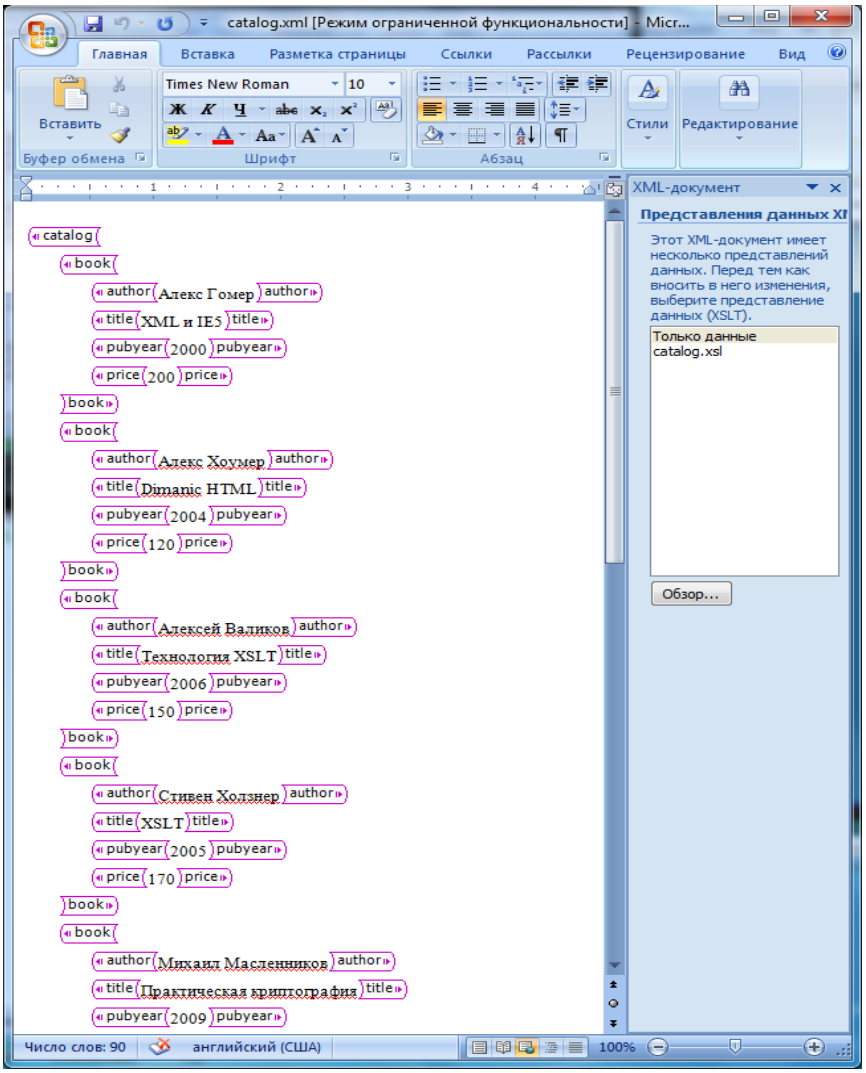
</xsl: for-each>

Контрольні питання.

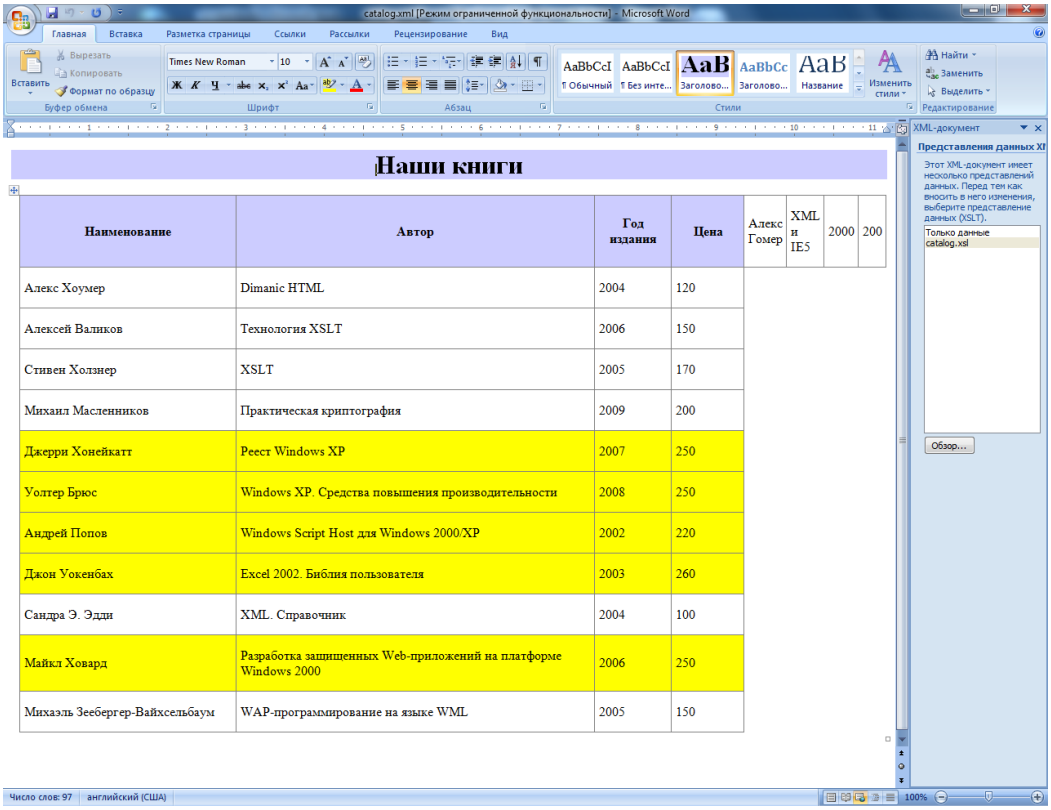
1. Які складові елементу xsl:template?
2. З якого елементу починається обхід дерева XML?
3. Які елементи вибирає xsl:apply-templates?
4. Які функції елементу xsl:value-of?
5. Якщо під час сортування наявні декілька ключів, то в якому порядку буде проводитися сортування?
6. Що таке дефолтне правило?
7. Швидкість обробки більша у звичайного шаблону чи у іменованого шаблону?

Тема 4. Інтеграція XML даних з MS Office

MS Office надає широкий спектр інструментів для роботи з XML-документами. Наприклад, відкриємо catalog.xml з використанням MS Word



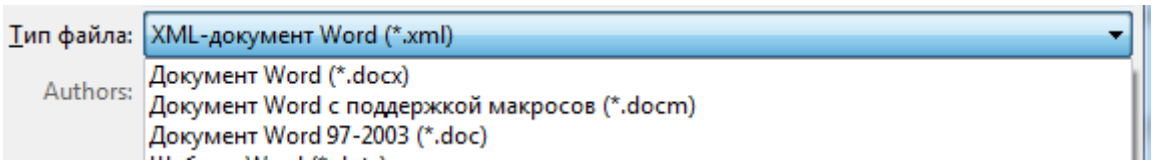
В результаті отримаємо відображення DOM-дерева даного документа. Якщо ж ми хоті побачити результат XSLT-перетворення, то, вибираючи в правому віконці файл з необхідним перетворенням, отримаємо відповідне відображення



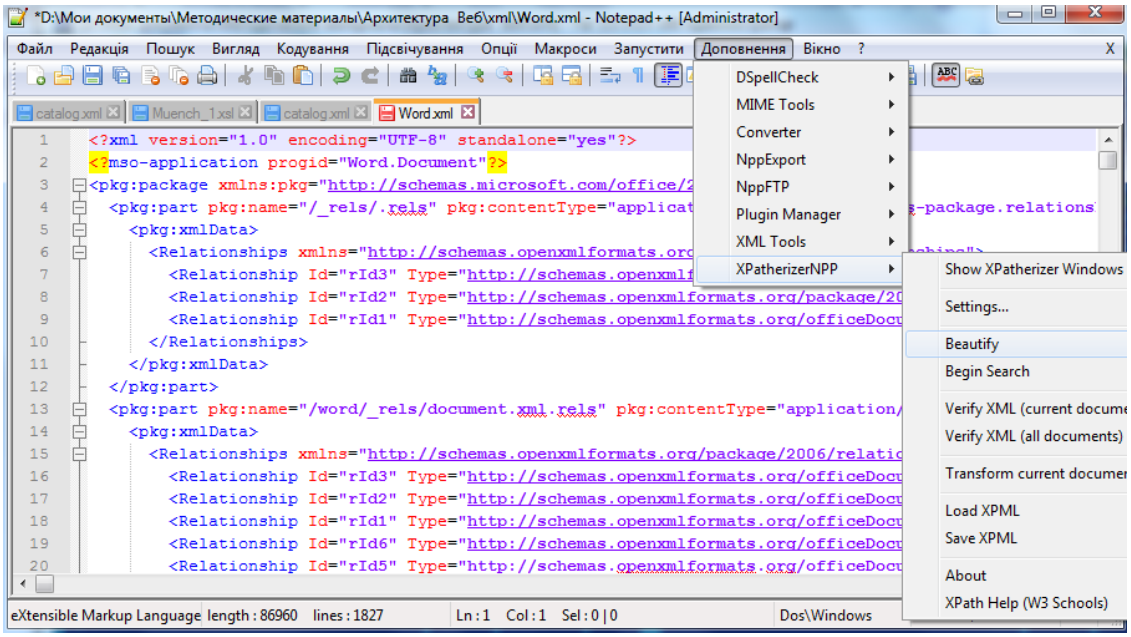
Наши книги

Наименование	Автор	Год издания	Цена	Алекс Гомер	XML и IE5	2000	200
Алекс Хоумер	Dimanic HTML	2004	120				
Алексей Валиков	Технология XSLT	2006	150				
Стивен Холзнер	XSLT	2005	170				
Михаил Масленников	Практическая криптография	2009	200				
Джерри Хонейкатт	Реест Windows XP	2007	250				
Уолтер Брюс	Windows XP. Средства повышения производительности	2008	250				
Андрей Попов	Windows Script Host для Windows 2000/XP	2002	220				
Джон Уокенбах	Excel 2002. Библия пользователя	2003	260				
Сандра Э. Эдди	XML. Справочник	2004	100				
Майкл Ховард	Разработка защищенных Web-приложений на платформе Windows 2000	2006	250				
Михаэль Зеебергер-Вайхсельбаум	WAP-программирование на языке WML	2005	150				

Цим можливості MS Word не обмежуються, хоча б помічаючи, що формат * .docx являє собою zip-контейнер XML-файла. Існує можливість збереження word-документа в нестислому вигляді -



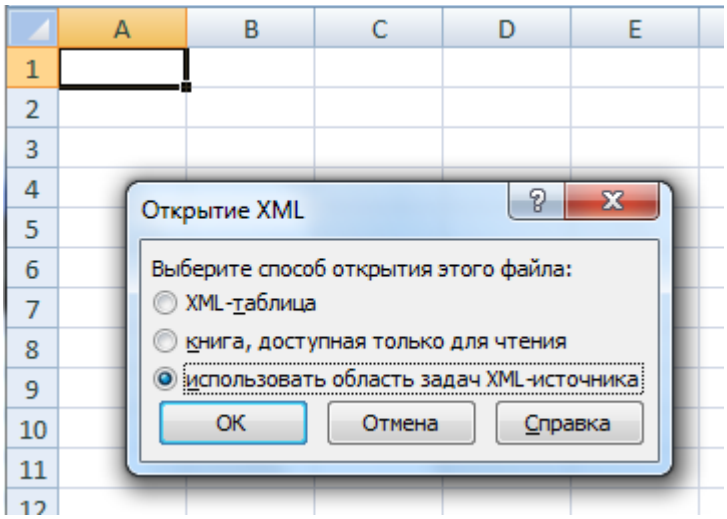
Для перегляду структури документа досить завантажити його в notepad ++.

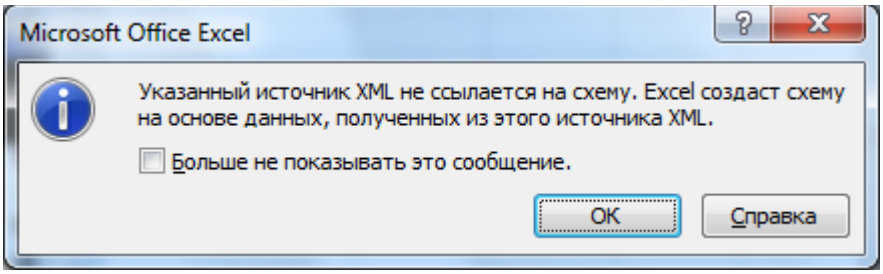


Таким чином, для створення word-документа з XML-документа, потрібно мати відповідне XSLT-перетворення в структуру WordML.

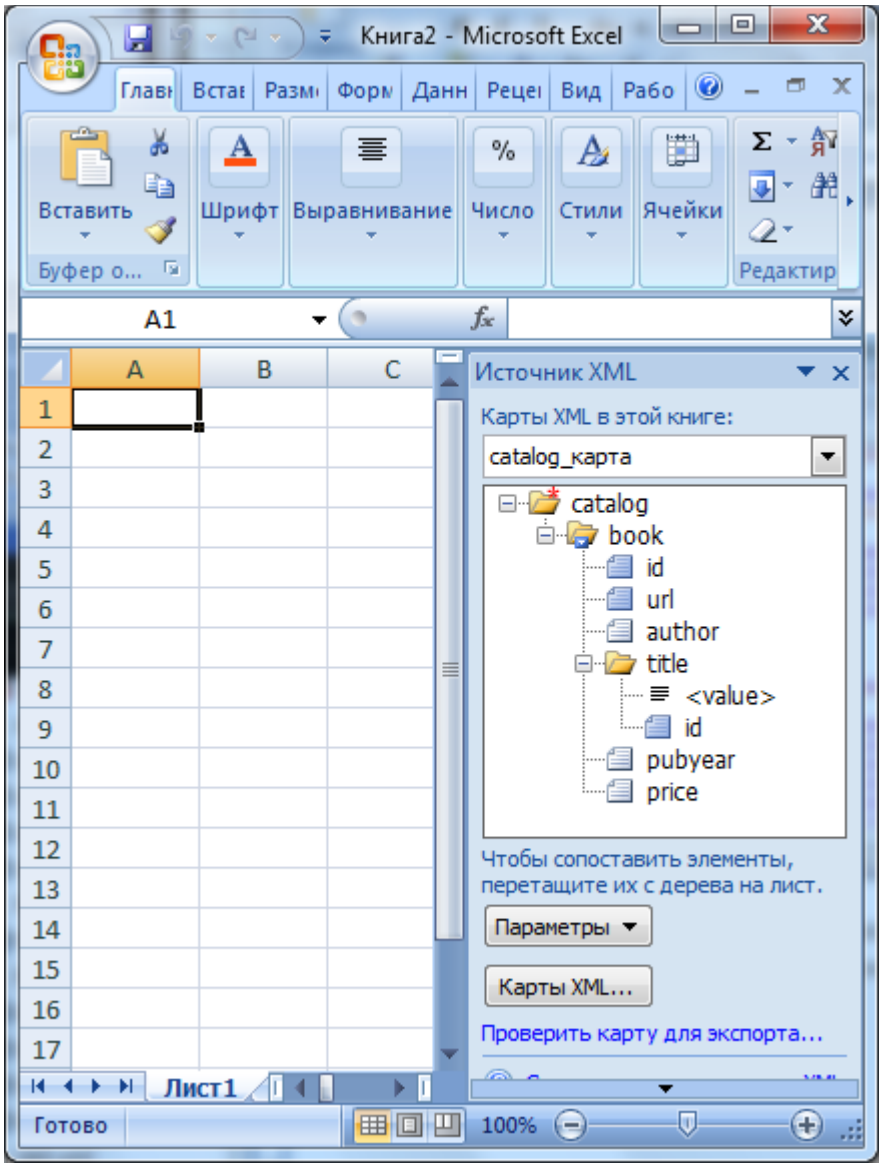
MS Excel

Відкриємо XML-документ засобами MS Excel

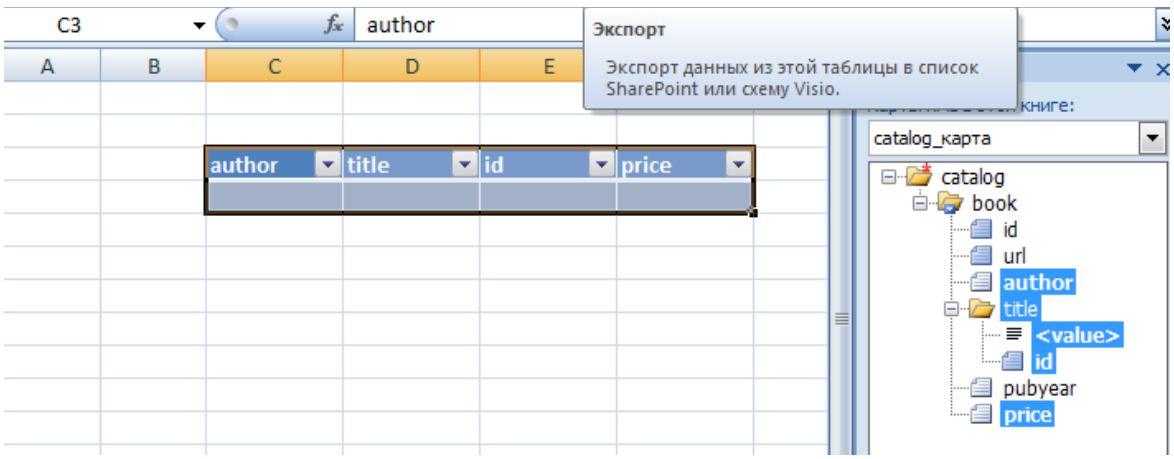




Після створення схеми маємо



Виділимо і перетягнемо на лист вузли, що нас цікавлять

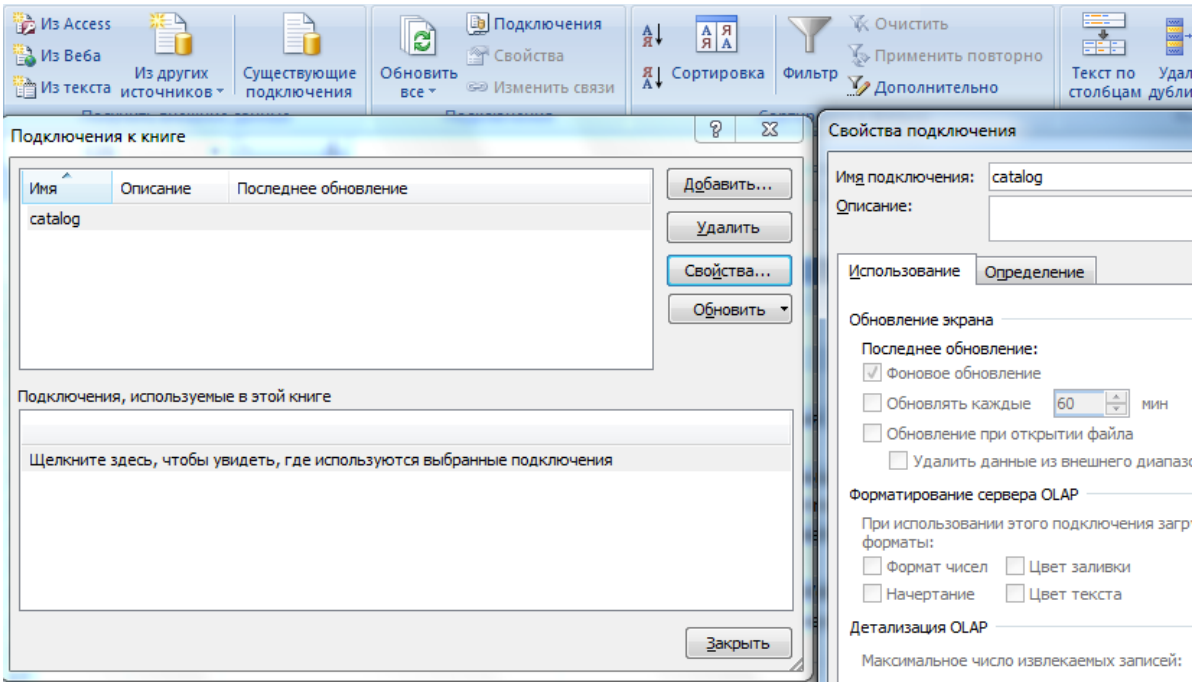


Натиснемо «оновити» і отримаємо таблицю

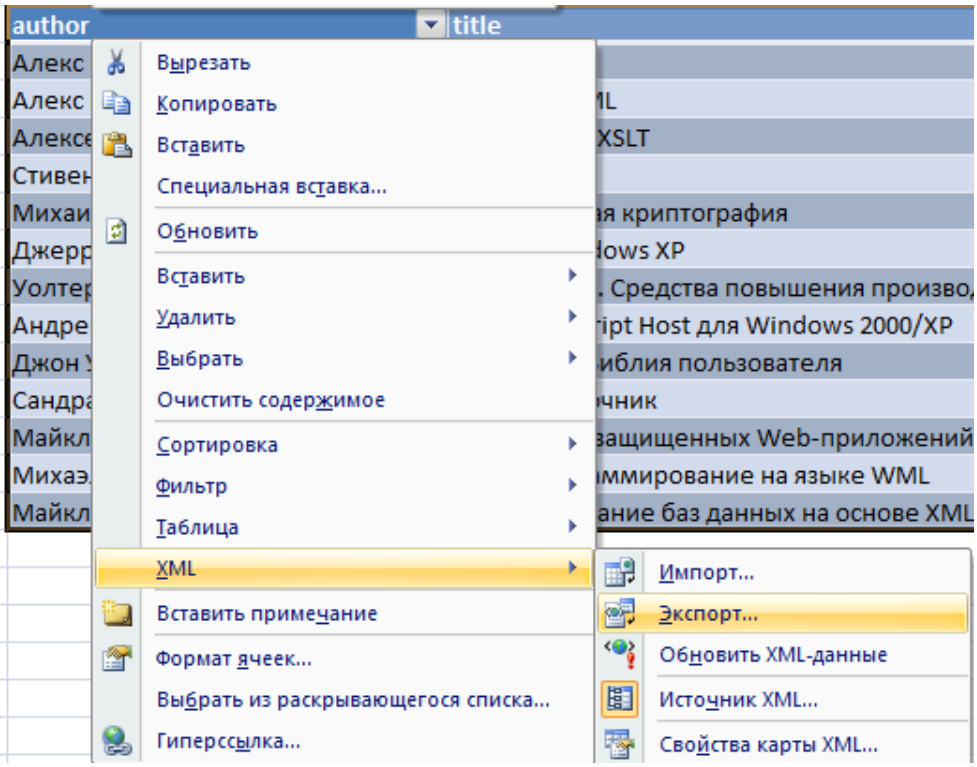
author	title	id	price
Алекс Гомер	XML и IE5	1	200
Алекс Гомер	Dimanic HTML	2	120

Алексей Валиков	Технология XSLT	3	150
Стивен Холзнер	XSLT		170
Михаил Масленников	Практическая криптография		200
Джерри Хонейкатт	Реестр Windows XP		250
Уолтер Брюс	Windows XP. Средства повышения производительности		250
Андрей Попов	Windows Script Host для Windows 2000/XP		220
Джон Уокенбах	Excel 2002. Библия пользователя		260
Сандра Э. Эдди	XML. Справочник		100
Майкл Ховард	Разработка защищенных Web-приложений на платформе Windows 2000		250
Михаэль Зеебергер-Вайхсельбаум	WAP-программирование на языке WML		150

Зазначисо, що в результаті ми провели не просто експорт, а «прив'язку», тобто, якщо змінити відповідний файл, наприклад, поміняти значення якогось вузла, то після обновлення зміниться і дана таблиця. Використовуючи закладку «підключення»

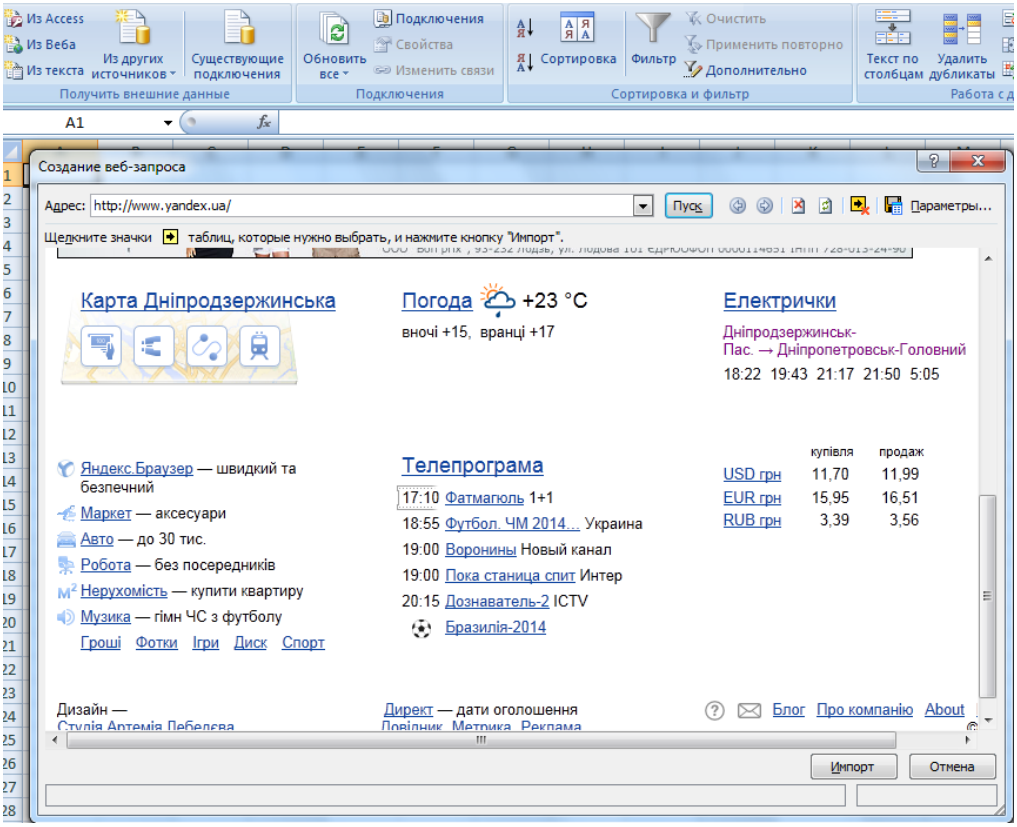


можна налаштувати автоматичне обновлення таблиці. А можна провести і звороттю оерацію – змінити таблицю exel. Виберемо з меню пункт «Експорт»

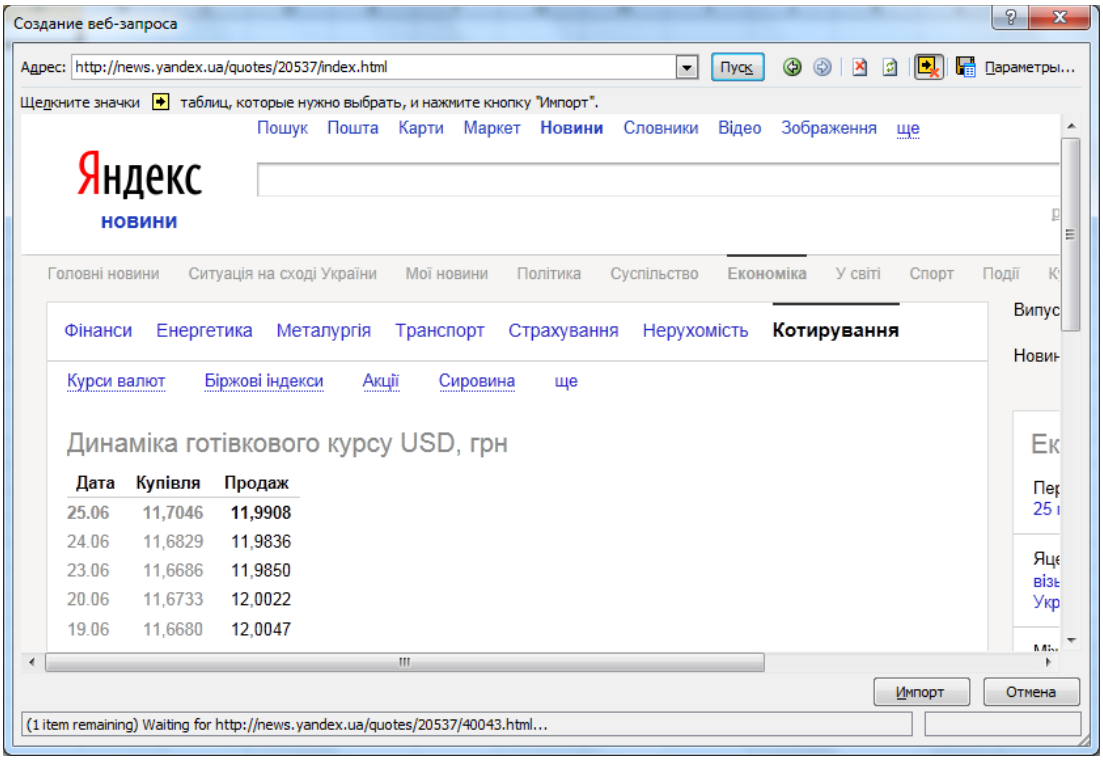


Ми отримаємо відповідний XML-документ.

Зауважимо, що зв'язок може бути встановлена і з html-документом. Приклад, на закладці «Дані» виберемо «з Веба»



Перейдемо до котирувань

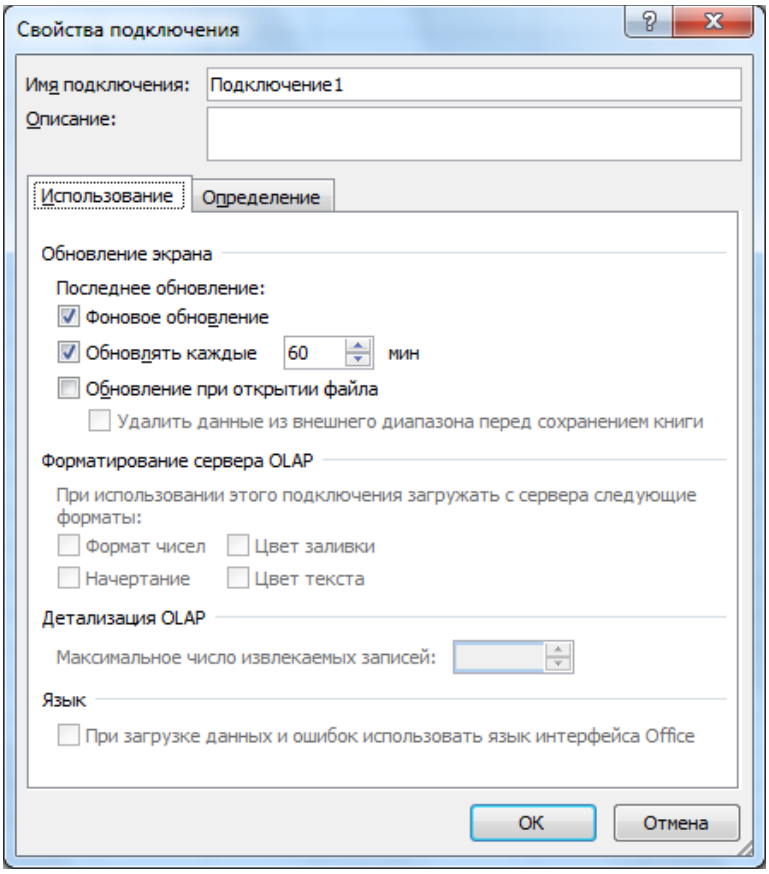


і натиснемо «Імпорт».

	A	B	C	D	E
1					
2		Курс валют			
3		Динаміка готівкового курсу USD, грн			
4		Дата	Купівля	Продаж	
5		25.Чер	11,7046	11,9908	
6		24.Чер	11,6829	11,9836	
7		23.Чер	11,6686	11,985	
8		20.Чер	11,6733	12,0022	
9		19.Чер	11,668	12,0047	
10		18.Чер	11,6407	11,9953	
11		17.Чер	11,594	11,9747	
12		16.Чер	11,59	11,9385	
13		13.Чер	11,495	11,855	
14		12.Чер	11,4392	11,8062	

Причому, при зміні курсу на відповідному WEB-сайті, буде змінюватися і таблиця. Зауважимо, значення цієї таблиці можна використовувати як звичайні значення excel-таблиці.

Як часто потрібно проводити оновлення залежить від вас



Excel також як і Word дозволяє зберігати дані в XML. Наприклад, зберігаючи таблицю в

«Таблиця XML 2003 (* .xml)»

	A	B	C
1	Шумейко О.О.	проф.	д.т.н.
2	Божуха Л.М.	доц.	к.ф.-м.н.
3	Бабенко М.В.	доц.	к.т.н.
4	Ялова К.М.	ст.викл.	к.т.н.

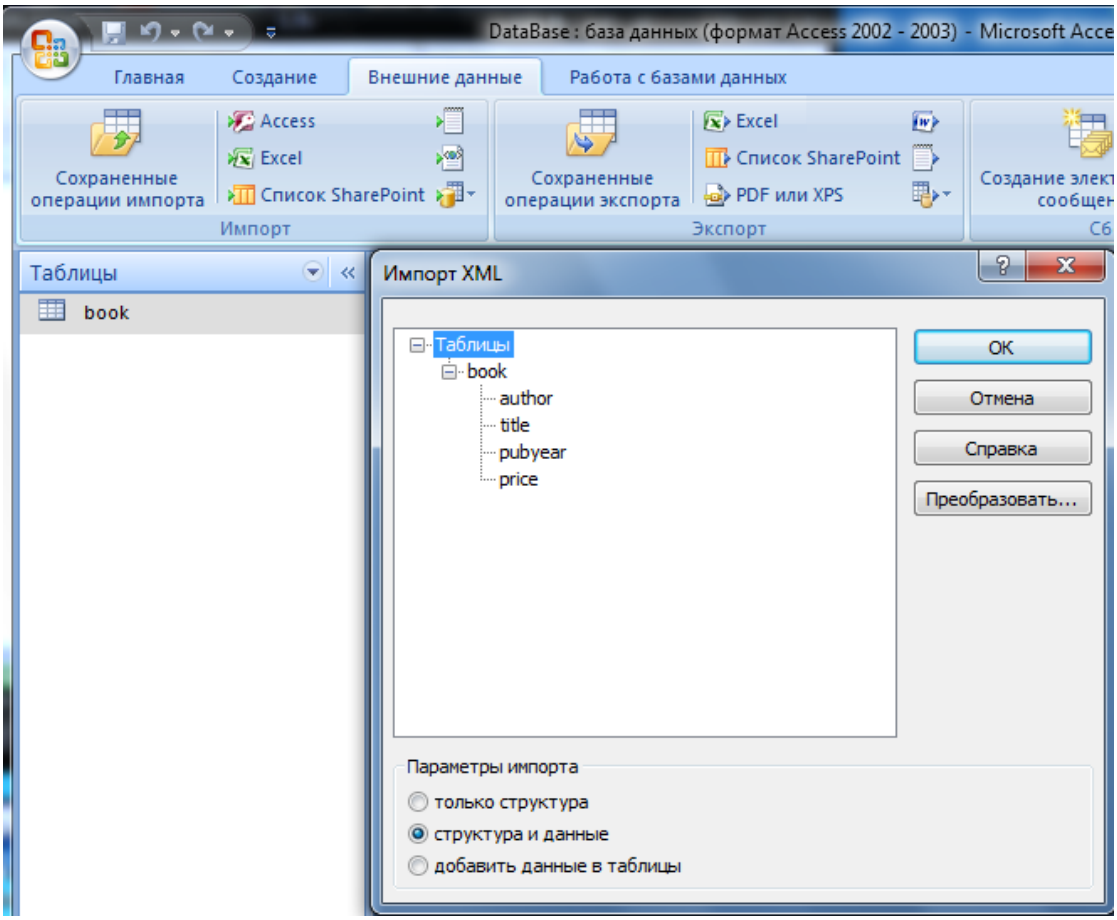
Отримуємо XML-документ

```
<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
……..
<Row>
   <Cell><Data ss:Type="String">Шумейко О.О.</Data></Cell>
   <Cell><Data ss:Type="String">проф.</Data></Cell>
   <Cell><Data ss:Type="String">д.т.н.</Data></Cell>
  </Row>
  <Row>
   <Cell><Data ss:Type="String">Божуха Л.М.</Data></Cell>
   <Cell><Data ss:Type="String">доц.</Data></Cell>
   <Cell><Data ss:Type="String">к.ф.-м.н.</Data></Cell>
  </Row>
  <Row>
```

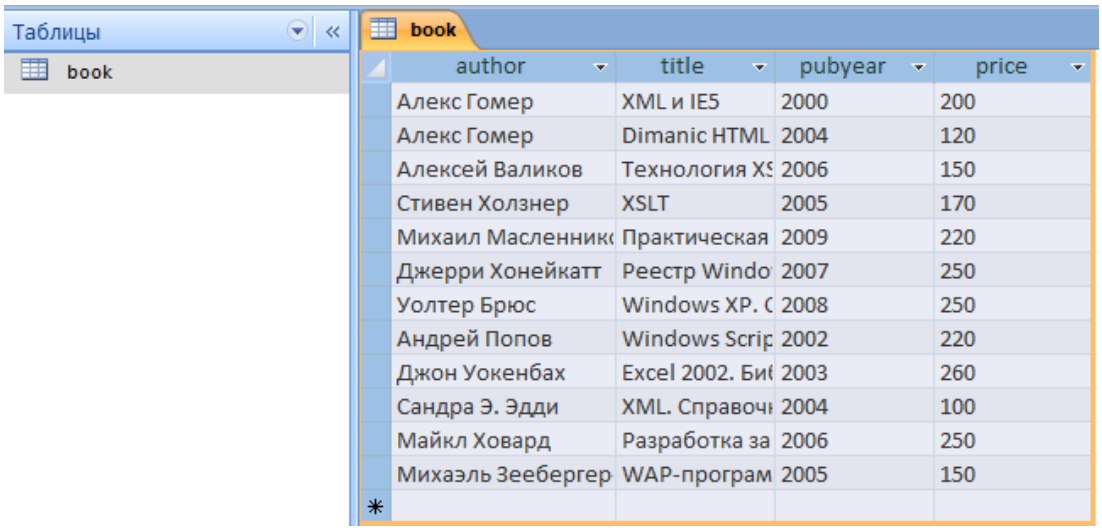
```
  <Cell><Data ss:Type="String">Бабенко М.В.</Data></Cell>
  <Cell><Data ss:Type="String">доц.</Data></Cell>
  <Cell><Data ss:Type="String">к.т.н.</Data></Cell>
 </Row>
 <Row>
  <Cell><Data ss:Type="String">Ялова К.М.</Data></Cell>
  <Cell><Data ss:Type="String">ст.викл.</Data></Cell>
  <Cell><Data ss:Type="String">к.т.н.</Data></Cell>
 </Row>
</Table>
```

MS Access

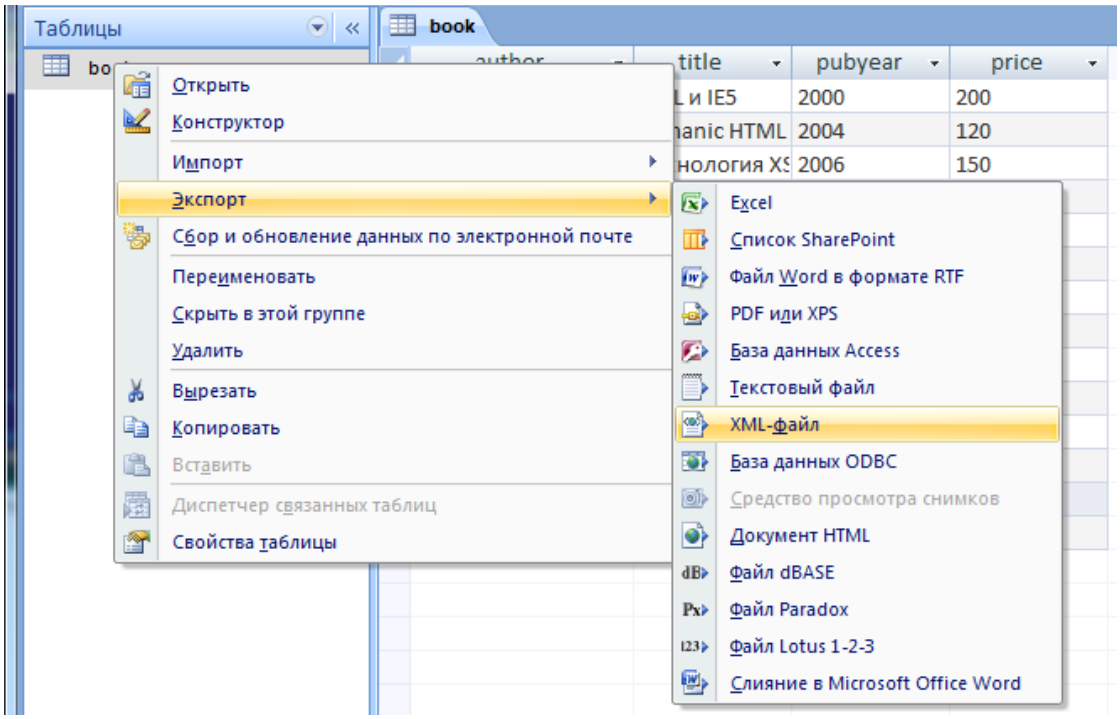
Цей додаток також дозволяє працювати з XML-файлами, проводячи як імпорт, так і експорт даних. Завантажимо MS Access і виберемо закладку «Зовнішні дані», натискаючи кнопку із зображенням Excel, виберемо XML-файл



і перетворимо його в таблицю



Можна внести зміни в дані і експортувати в XML-файл



Причому, можна не просто зберегти дані у вигляді XML, а й отримати відповідну XML-схему і навіть XSLT-перетворення в HTML.

MS InfoPath

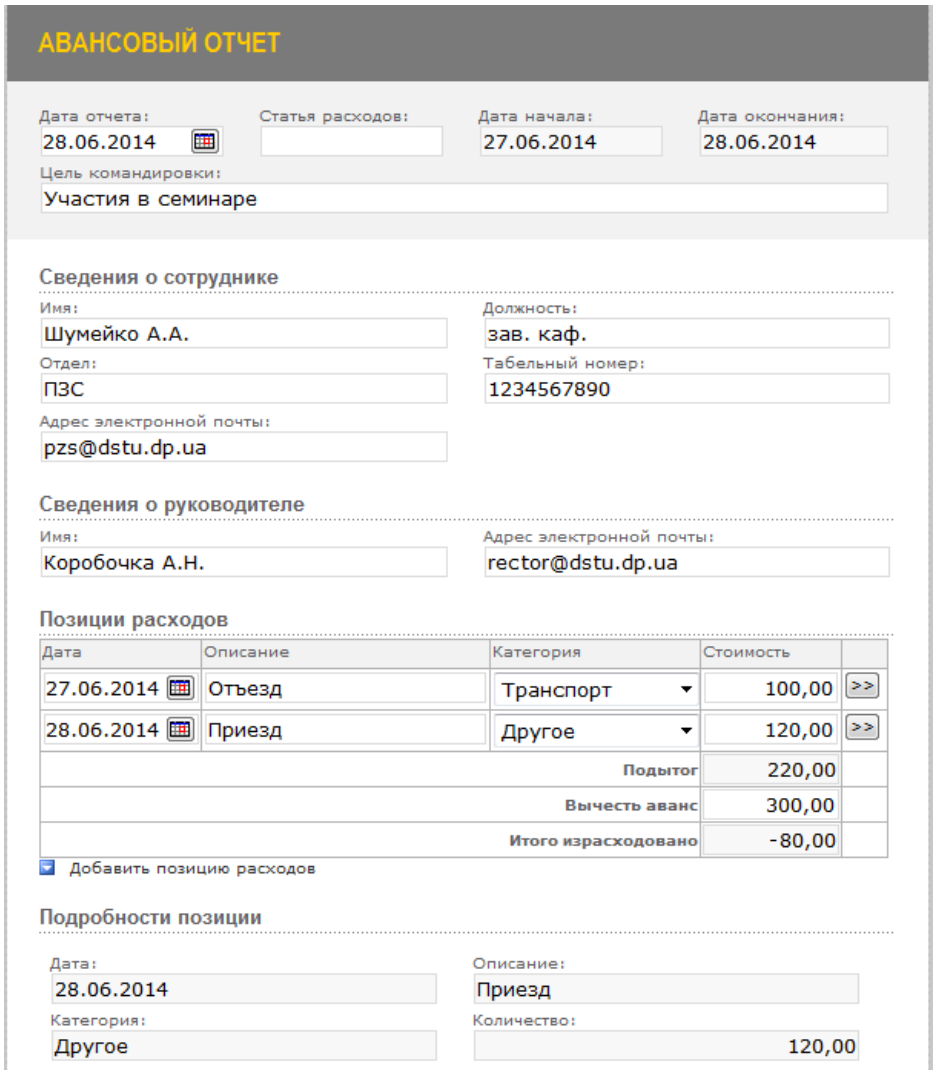
MS InfoPath, на відміну від розглянутих додатків, створений для роботи з XML. Цей додаток вперше з'явився в 2003 році і від версії до версії отримує активний розвиток. На відміну від розглянутих раніше додатків, MS InfoPath є бізнес-додатком. Ідеологія MS InfoPath полягає в тому, що досить часто доводиться працювати з формами (тобто документами з строгою структурою, тобто тип даних і місця їх розташування жорстко зафіксовані).

MS InfoPath дозволяє

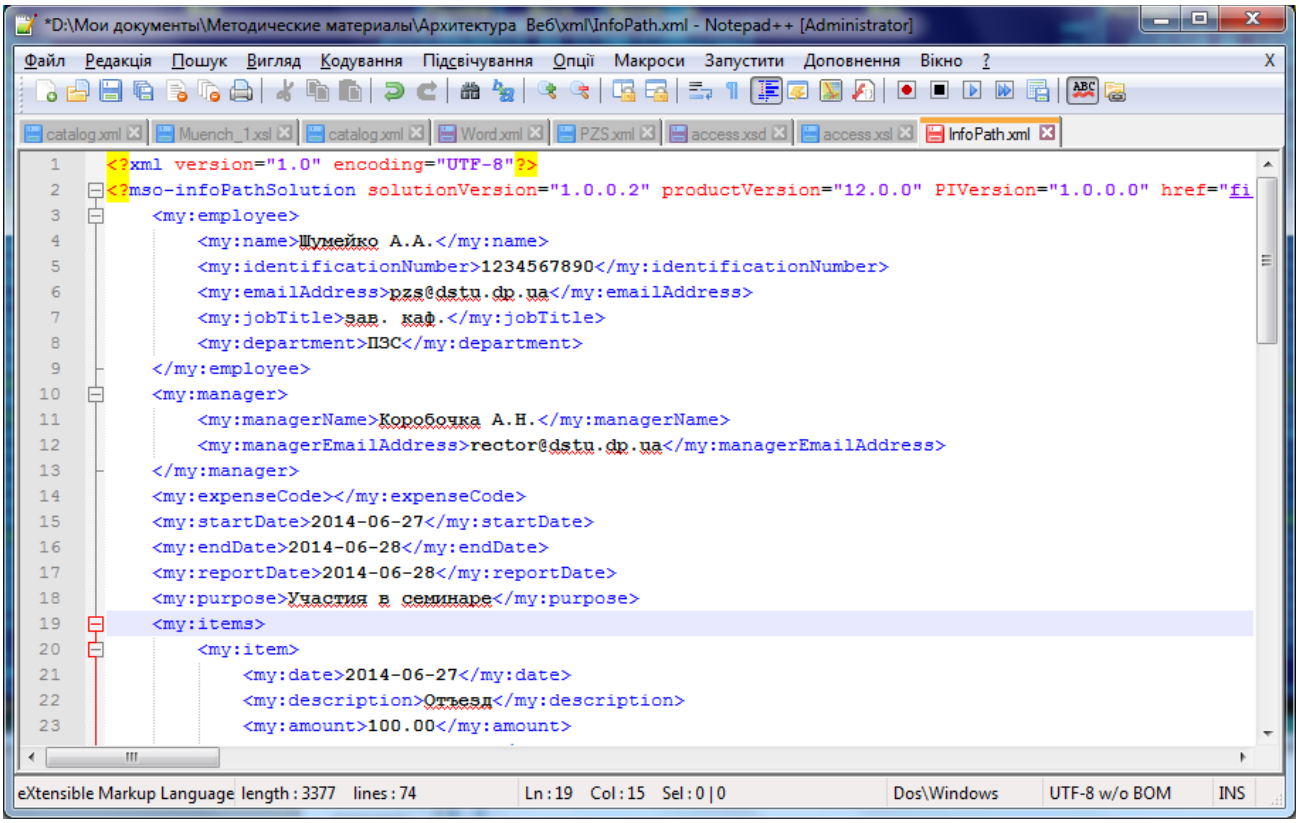
- Швидко створювати різноманітні форми,
- Реалізує повторне використання форм (формат даних - XML),
- Легко збирає колекцію (бібліотеки форм MS InfoPath),
- Можна для роботи не вимагати сам додаток MS InfoPath -Forms Server,
- Легко інтегрується в існуючі бізнес-процеси.

Як приклад, якщо після завантаження MS InfoPath завантажити шаблон «авансовий звіт», то після збереження на диск отримаємо cab-контейнер. Даний файл повинен лежати на загальнодоступному місці в режимі «тільки для читання».

Викличемо до життя цей файл і заповнимо його

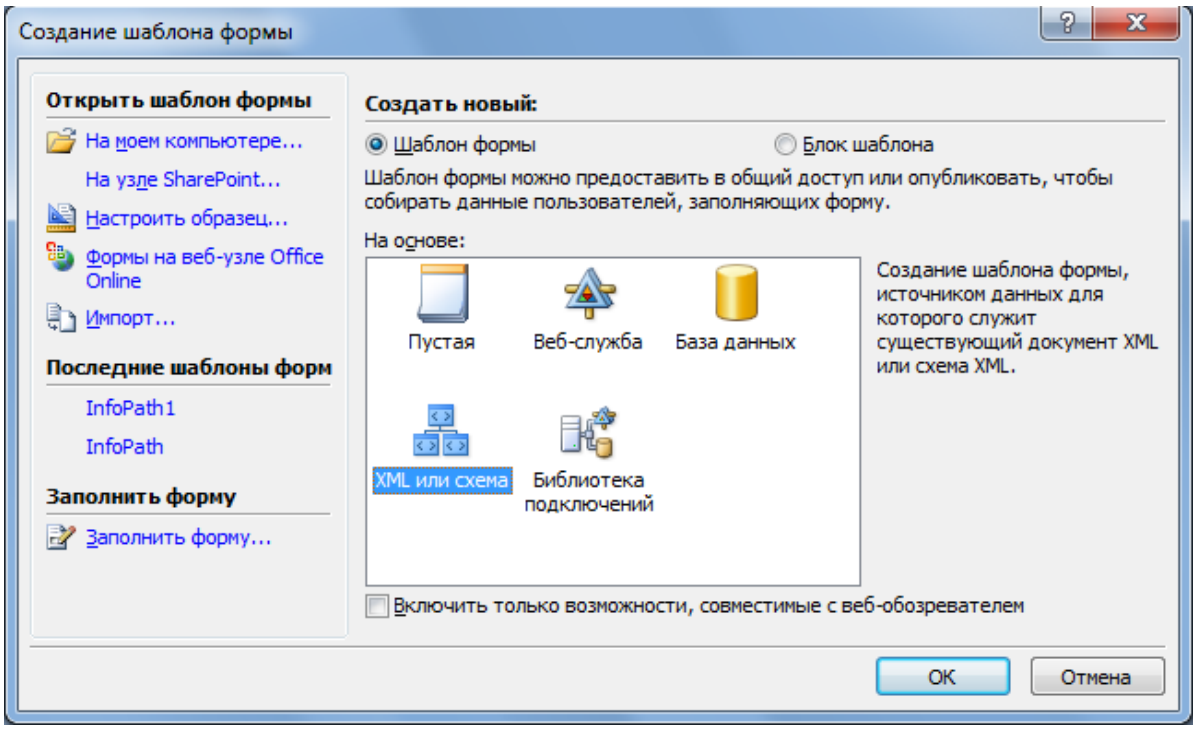


Після збереження отримаємо «чистий» XML-документ.

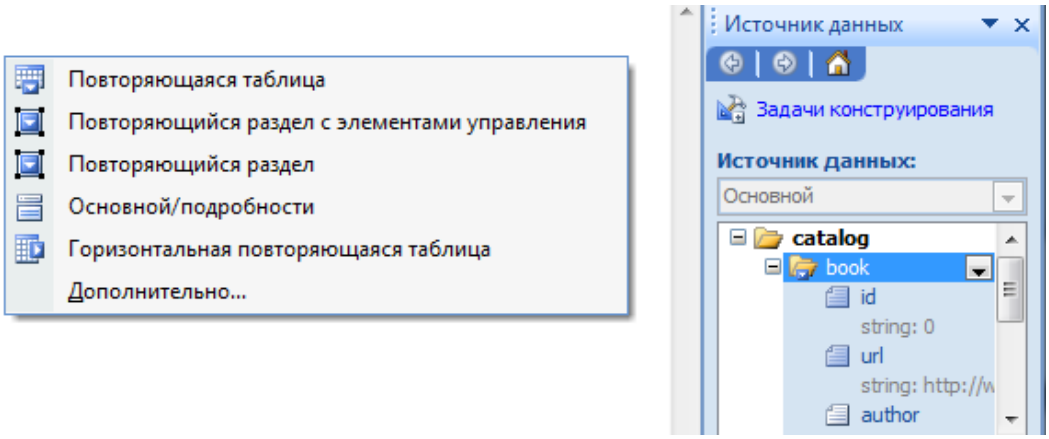


Природно, можна створювати нові форми, їх версії для друку, встановлювати перевірки осередків або навіть програмувати їх.

Відкриємо MS InfoPath в режимі конструктора і створимо новий шаблон форм і зробимо його на основі існуючого XML.



Виберемо необхідний файл і перетягнемо на поле MS InfoPath

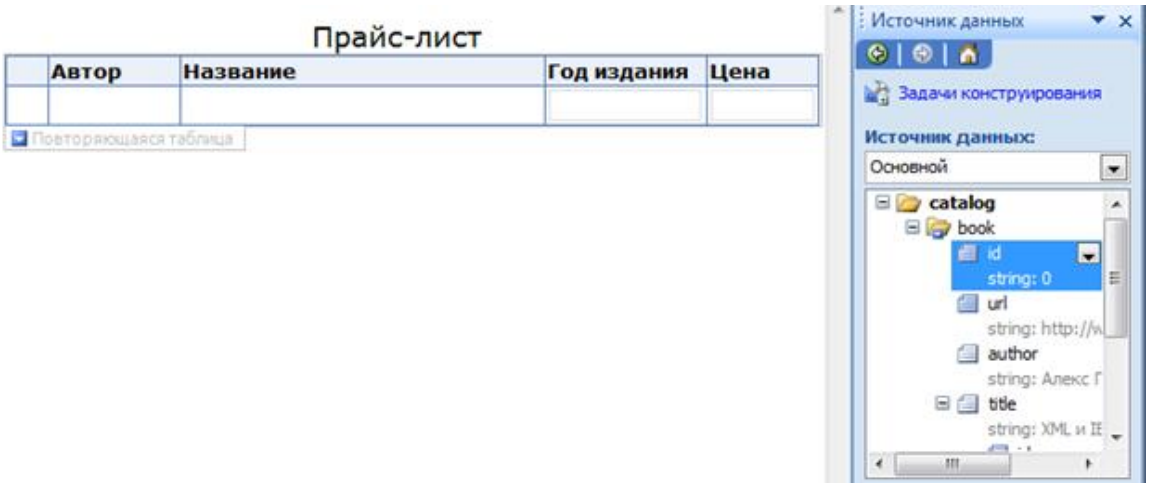


Вибираючи таблицю, отримаємо

Прайс-лист

Id	Url	Автор	Название	Год	Цена
0	http://www.homer.com	Алекс Гомер	XML и IE5	2004	200

Можна для однієї форми робити кілька подань - створити нову форму, перетягнути на неї з джерела даних той же наш XML і створити форму для друку



Файли MS InfoPath мають розширення * .xsn і являють собою cab-контейнер, для нашого прикладу він складається з наступних файлів

..
Catalog.xsd
Manifest.xsf
Sampledata.xml
Schema.xsd
Template.xml
upgrade.xsl
view1.xsl
view2.xsl

Контрольні питання.

1. Які програми MS Office підтримують XML?
2. Чи є документ *.doc XML-документом?
3. Чи є документ *.docx XML-документом?
4. Яка програма MS Office спеціалізована для роботи з XML?

Тема 5. Стандарт XML-RPC

XML-RPC (Extensible Markup Language Remote Procedure Call - XML-виклик віддалених процедур) - стандарт/протокол виклику віддалених процедур, заснований на XML.

Типовий приклад запиту XML-RPC:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
        <param>
              <value><i4>41</i4></value>
        </param>
  </params>
</methodCall>
```

Типовий приклад відповіді на запит XML-RPC:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
        <param>
              <value><string>South Dakota</string></value>
        </param>
  </params>
</methodResponse>
```

Протокол XML-RPC був спочатку розроблений Дейвом Вінером з компанії «UserLand Software» у співпраці з Майкрософт, в 1998 році. Однак корпорація Майкрософт незабаром визнала цей протокол занадто спрощеним, і почала розширювати його функціональність. Після декількох циклів по розширенню функціональності, з'явилася система, нині відома як SOAP. Пізніше Майкрософт почала широко рекламувати і впроваджувати SOAP, а початковий XML-RPC був відкинутий. Але, незважаючи на відкидання його Майкрософтом, стандарт XML-RPC зачарував багатьох програмістів своєю надзвичайною простотою і, за рахунок цього, існує донині і навіть поступово набирає популярність.

Типи даних

Ім'я типу	Приклад тегу	Опис типу
array	<**array**> <**data**> <**value**><**i4**>1404</**i4**></**value**> <**value**><**string**>Что-нибудь здесь</**string**></**value**> <**value**><**i4**>1</**i4**></**value**> </**data**> </**array**>	Масив величин, без ключів
base64	<**base64**> eW91IGNhbid0IHJlYWQgdGhpcyE= </**base64**>	Закодовані в Base64 бінарні дані
boolean	<**boolean**>1</**boolean**>	Логічна (булева) величина (0 чи 1)
date/time	<**dateTime.iso8601**>20140717T14:08:55 </**dateTime.iso8601**>	Дата і час
double	<**double**>-12.53</**double**>	Дробная величина подвоєної точності
integer	<**i4**>42</**i4**>	Ціле число
string	<**string**>Здравствуй, Мир!</**string**>	Строка символів (в тій же кодіровці, що і весь XML-документ)
struct	<**struct**> <**member**> <**name**>Что-то</**name**> <**value**><**i4**>1</**i4**></**value**> </**member**> <**member**> <**name**>Ещё что-то</**name**> <**value**><**i4**>2</**i4**></**value**> </**member**> </**struct**>	Масив величин, з ключами
nil	<**nil**/>	Нульова (пуста) величина — це расширення XML-RPC

При цьому теги з типом даних при передачі від клієнта можуть бути опущені і не вказуватися зовсім.

XML-RPC функції PHP (http://php.net/manual/ru/ref.xmlrpc.php)

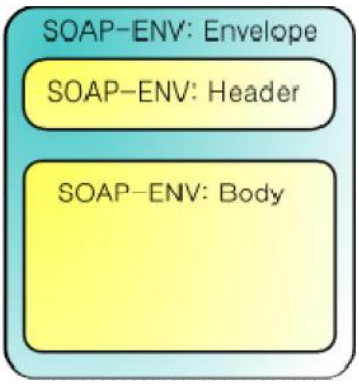
- xmlrpc_decode_request - Декодує XML в нативні типи PHP
- xmlrpc_decode - Декодує XML в нативні типи PHP
- xmlrpc_encode_request - Генерує XML для методу запиту
- xmlrpc_encode - Генерує XML для PHP значення
- xmlrpc_get_type - Отримує XML-RPC тип для значення PHP

- xmlrpc_is_fault - Визначає, чи є масив значень описупомилки XMLRPC
- xmlrpc_parse_method_descriptions - Декодує XML в список описів методів
- xmlrpc_server_add_introspection_data - Додає документацію самоаналізу
- xmlrpc_server_call_method - Розбирає XML запити і викликає методи
- xmlrpc_server_create - Створює XML-RPC сервер
- xmlrpc_server_destroy - Знищує серверні ресурси
- xmlrpc_server_register_introspection_callback - Реєструє функцію PHP для генерації документації
- xmlrpc_server_register_method - Реєструє функцію PHP до методу ресурсу, відповідному method_name
- xmlrpc_set_type - Встановлює тип XML-RPC, base64 або datetime для значення рядка PHP

SOAP

SOAP (Simple Object Access Protocol - простий протокол доступу до об'єктів) - протокол обміну структурованими повідомленнями в розподіленому обчислювальному середовищі. Спочатку SOAP призначався в основному для реалізації віддаленого виклику процедур (RPC). Зараз протокол використовується для обміну довільними повідомленнями у форматі XML, а не тільки для виклику процедур. SOAP є розширенням протоколу XML-RPC.

Структура SOAP-пакету:



Приклад SOAP - запиту:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
        <getProductDetails xmlns="http://warehouse.example.com/ws">
              <productID>12345</productID>
        </getProductDetails>
      </soap:Body>
</soap:Envelope>
```

Приклад SOAP - відповіді:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
        <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productID>12345</productID>
```

```
        <productName>Стакан граненый</productName>
        <description>Стакан граненый. 250 мл.</description>
        <price>9.95</price>
        <inStock>true</inStock>
      </getProductDetailsResult>
</getProductDetailsResponse>
</soap:Body>
</soap:Envelope>
```

WSDL

WSDL (Web Services Description Language) - мова опису веб-сервісів, заснована на мові XML.

Кожен документ WSDL можна розбити на три логічні частини:

1. Визначення типів даних - визначення виду даних, що відправляються і одержуються сервісом XML повідомлень
2. Абстрактні операції - список операцій, які можуть бути виконані з повідомленнями
3. Зв'язування сервісів - спосіб, яким повідомлення буде доставлено

Робота з SOAP

1. **Клієнтський додаток**

Створення екземпляра класу:

```
$client = new SoapClient("sms.wsdl", array(
'soap_version'  => SOAP_1_2,  // версія SOAP
'login'  => "sms_name",   // HTTP авторизація
'password'  => "sms_password",
'proxy_host'  => "localhost",  // Налаштування proxy
'proxy_port'  => 8080,
'proxy_login'  => "sms_name",
'proxy_password'=> "sms_password",
'encoding'  => 'Windows-1251', // кодування вмісту (але не
самого SOAP-пакету!)
'trace'  => true  // трасування запиту
));
```

Функції:

$client -> __ getFunctions() // Отримати список доступних функцій

$client -> __ getLastRequest() // Отримати XML останнього запиту

$client-> __setLocation ('http://www.somethirdparty.com') // Встановити нову адресу для запиту

$ Client -> __ getLastRequest() // Отримати останній запиту

$client ->__getLastRequestHeaders()//Отримати заголовок останнього запиту

$client -> __ getLastResponse() // Отримати останній відповідь

$client->__getLastResponseHeaders()//Отримати заголовок останньої відповіді

$client -> __ getTypes() // отримання типів даних, оголошених в wsdl

$client->__setCookie ('cookie_name', 'cookie_value') // Установка cookie

$client -> __ soapCall ("getStatus", array (50996) // Викликати функцію

// Установка заголовків

$headers = array();

$headers [] = new SoapHeader ('http://soapinterop.org/echoheader/',

'echoMeStringRequest', 'hello world');

$headers [] = new SoapHeader ('http://soapinterop.org/echoheader/',

'echoMeStringRequest', 'hello world again');

$client -> __ setSoapHeaders ($ headers);

2. Серверний додаток

Створення екземпляра класу:

$server = new SoapServer ("sms.wsdl", array ('classmap' =>

array ('mixed' => "MyMixed")));

Функції:

// Додати функцію сервера

$server-> addFunction ("test");

$server-> handle();

$server-> setClass ("SoapService"); // Встановити клас

$server-> setObject (new SoapService()); // Встановити об'єкт

$server-> fault (404, 'WEBGURU NOT FOUND!'); // Повернути повідомлення про помилку

Приклади реалізації SOAP-сервера і клієнта (відсилання SMS).

sms.wsdl

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:tns="http://litesms.net/sms_soap.php"
                xmlns:soap-env="http://schemas.xmlsoap.org/wsdl/soap/"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                targetNamespace="http://litesms.net/sms_soap.php">
<types>
// XSD-схема. Опис змінних
```

```
<xsd:schema>
        <xsd:complexType name="array">
                <xsd:complexContent>
                <xsd:restriction base="soapenc:array">
                        <xsd:attribute ref="soapenc:arrayType"
                        arrayType="tns:mixed[]"/>
                </xsd:restriction>
                </xsd:complexContent>
        </xsd:complexType>
        <complexType name="mixed">
                <all>
                        <element name="varString" type="xsd:string"/>
                        <element name="varInt" type="xsd:int"/>
                        <element name="varFloat" type="xsd:float"/>
                </all>
        </complexType>
      </xsd:schema>
</types>
// Далі йде опис параметрів сервісу (вхідні або вихідні)

<message name="sendSmsInput">
      <part name="message" type="xsd:string"/>
      <part name="to" type="xsd:string"/>
      <part name="from" type="xsd:string"/>
</message>
<message name="getBalanceInput"/>
<message name="getStatusInput">
      <part name="message_id" type="xsd:int"/>
</message>
<message name="sendSmsOutput">
      <part name="return" type="xsd:string"/>
</message>
<message name="getBalanceOutput">
      <part name="return" type="xsd:string"/>
</message>
<message name="getStatusOutput">
      <part name="return" type="xsd:string"/>
</message>
<message name="testInput">
      <part name="return" type="tns:mixed"/>
</message>
<message name="testOutput">
      <part name="return" type="tns:mixed"/>
</message>
// Операції. Опис можливостей, що надаються WEB-сервісом, які описуються визначеними раніше параметрами

<portType name="AvistSmsServerPortType">
      <operation name="sendSms">
        <input message="tns:sendSmsInput"/>
        <output message="tns:sendSmsOutput"/>
```

```
    </operation>
    <operation name="getBalance">
      <input message="tns:getBalanceInput"/>
      <output message="tns:getBalanceOutput"/>
    </operation><operation name="getStatus">
      <input message="tns:getStatusInput"/>
      <output message="tns:getStatusOutput"/>
    </operation>
    <operation name="test">
      <input message="tns:testInput"/>
      <output message="tns:testOutput"/>
    </operation>
</portType>
// Опис формату даних

<binding name="AvistSmsServerBinding" type="tns:AvistSmsServerPortType">
    <soap-env:binding xmlns="http://schemas.xmlsoap.org/wsdl/soap/" style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/"
name="sendSms">
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
    <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/"
name="getBalance">
// Назва операції і далі - параметри

<input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
    <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/"
name="getStatus">
      <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
        </output>
      </operation>
      <operation xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/" name="test">
        <input xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
              <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/">
              <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output>
      </operation>
</binding>
<service name="AvistSmsServerService">
// Визначення точки входу самого сервісу

<port xmlns:default="http://schemas.xmlsoap.org/wsdl/soap/"
      name="AvistSmsServerPort" binding="tns:AvistSmsServerBinding">
        <soap-env:address xmlns="http://schemas.xmlsoap.org/wsdl/soap/"
        location="http://soapserver/soap_server.php"/>
// Тут прописаний адреса SOAP-сервера. Сюди і відправляється запит.

</port>
</service>
</definitions>
```

Для використання SOAP потрібно в php/ext мати бібліотечку php_soap.dll і підключити її в php.ini.

Soap-клієнт має вигляд

soap_send_sms.php

```
<?

require_once ('creds.php');

ini_set ("soap.wsdl_cache_enabled", "1"); // Включаємо кешування WSDL

// Створюємо SOAP-клієнта

$client = new SoapClient ("http://litesms.net/sms_soap.php?wsdl",

array ('login' => $ login,

'Password' => $ password,

'Trace' => true

// Трасування запиту

));

try

{
```

```
// Виводимо всі функції
//print_r ($ client -> __ getFunctions());
echo '<br />';
// Відправляємо SMS
//echo $ client-> sendSMS ("Privet, WebGuru!", "79085018676", "prog-school.ru"). '<br />';
// Установка нової адреси для запитів
//$client->__setLocation('http://www.somethirdparty.com ');
// Отримуємо баланс в системі
//echo $ client-> getBalance(). '<br />';
// Отримуємо статус повідомлення
echo $ client-> getStatus (50996). '<br />';
// var_dump ($ client -> __ getLastRequest());
// var_dump ($ client -> __ getLastRequestHeaders());
// var_dump ($ client -> __ getLastResponse());
// var_dump ($ client -> __ getLastResponseHeaders());
// Отримання типів даних, оголошених в wsdl
// var_dump ($ client -> __ getTypes());
// Установка cookie
// $client -> __ setCookie ('cookie_name', 'cookie_value');
// Установка SOAP заголовків
/ *
$headers = array();
$headers [] = new SoapHeader ('http://soapinterop.org/echoheader/',
'echoMeStringRequest',
'Hello world');
$headers [] = new SoapHeader ('http://soapinterop.org/echoheader/',
'echoMeStringRequest',
'Hello world again');
$client -> __ setSoapHeaders ($ headers);
```

```
$client-> getStatus (50996). '<br />';
var_dump ($ client -> __ getLastRequest());
* /
// Виклик функції
var_dump ($ client -> __ soapCall ("getStatus", array (50996)));
}
catch (SoapFault $ exception) {
echo $ exception;
}
?>
```

де creds.php

```
<?php
        $login = 'your_login';
        $password = 'your_pass';
?>
```

Soap-сервер.

```
<?
class MyMixed {
   public $varString;
   public $varFloat;
   public $varInt;
}
// Ці типи даних (назва збігається) описані в комплексному типі mixed відповідного файлу wsdl
$server = new SoapServer ("sms.wsdl", array ('classmap' => ('mixed' => "MyMixed")));
// Тут зв'язок між даними wsdl і функціями сервера
// Тип mixed визначений у клієнті
ini_set ("soap.wsdl_cache_enabled", "0");
// Відключаємо кешування WSDL, щоб мати актуальну версію
function test (MyMixed $ data) {
```

```
return ($ data-> varString);
}
$ Server-> addFunction ("test");
```

// Операція test визначена в wsdl, а аналогічна функція визначена в серверному скрипті (вище) з тією ж назвою

```
$ Server-> handle();
```

// Обробка запиту з упаковкою відповіді в формат soap.

```
?>
```

і, власне кажучи, скрипт обробки клієнтського запиту

soap_client.php

```
<?
header ('Content-Type: text/html; charset = utf-8');
$client = new SoapClient ("sms.wsdl");
$client-> __setLocation ('http: //soapserver/soap_server.php');
//$client->__setLocation('http://soapserver/soap_server2.php ');
```

// __ SetLocation дозволяє переписати точку доступу, яка визначена в wsdl

```
class MyMixed
{
   public $ varString;
   public $ varFloat;
public $ varInt;
}
$param = new MyMixed();
$param-> varString = "Рядок";
$param-> varFloat = 3.5;
$param-> varInt = 45;
try
{
var_dump ($ client-> test ($ param));
} Catch (SoapFault $ exception)
```

```
{
var_dump ($ exception);
}
?>
```

Виклик цієї функції дасть

```
string (12) "Рядок"
```

Відпрацювала функція сервера і повернула значення varString (див. Серверний скрипт).

Розглянемо використання ООП у роботі серверного скрипта

soap_server2.php

```
<?
class MyMixed {
   public $ varString;
   public $ varFloat;
public $ varInt;
}
class SoapService {
private $ __ myMixedInstance;
function changeVarString() {
$this -> __ myMixedInstance-> varString = "Новий рядок!";
return $ this -> __ myMixedInstance;
// Полю varString встановлюється нове значення
}
function test (MyMixed $ data) {
$this -> __ myMixedInstance = $ data;
return $this-> changeVarString();
// Global $ server;
// $server-> addSoapHeader (new
SoapHeader ('http://soapinterop.org/echoheader/',
        // 'SoapHeaderKey',
```

```
        // 'SoapHeaderValue'));
// Return $server-> getFunctions();
}
}
// Відключаємо кешування WSDL
ini_set ("soap.wsdl_cache_enabled", "0");
$server = new SoapServer ("sms.wsdl",
array ('classmap' =>
array ('mixed' => "MyMixed")));
// $server-> setClass ("SoapService");
$server-> setObject (new SoapService());
$server-> handle();
// $server-> fault (404, 'WEB-SERVICE NOT FOUND!');
?>
```

Повернення класу проводиться за допомогою setClass із зазначенням конкретного класу, в якому визначені всі необхідні властивості й методи.

У цьому випадку повертається

```
object (stdClass) # 3 (3) {
["varString"] =>
string (24) "Новий рядок!"
["varFloat"] =>
float (3.5)
["varInt"] =>
int (45)
}
```

тобто отримуємо об'єкт.

Контрольні питання.

1. Які функції запиту XML-RPC?
2. Що описує мова WSDL?
3. В чому мета і функції протоколу SOAP?

Тема 6. Формат JSON

JSON (JavaScript Object Notation) - текстовий формат обміну даними, заснований на JavaScript і зазвичай використовуваний саме з цією мовою. JSON працює тільки з кодуванням UTF-8.

Порівняння форматів XML і JSON:

XML:

```
<person>
  <firstName>Петро</firstName>
  <lastName>Петренко</lastName>
  <address>
        <streetAddress>Дніпробудівська, 2а, кв.314</streetAddress>
        <city>Дніпродзержинськ</city>
        <postalCode>51917</postalCode>
  </address>
  <phoneNumbers>
        <phoneNumber>805692-3-45-67</phoneNumber>
        <phoneNumber>805692-3-45-68</phoneNumber>
  </phoneNumbers>
</person>
```

JSON:

```
  firstName: "Петро",
  lastName: "Петренко",
  address: {
  streetAddress: " Дніпробудівська, 2а, кв.314",
  city: "Дніпродзержинськ",
  postalCode: 51917
  },
  phoneNumbers: [
  "805692-3-45-67",
  "805692-3-45-68"
  ]
  //Масив[,]
```

Дані в цьому форматі задаються у вигляді масиву - асоціативного (пара ім'я: значення) або перенумерованого і приймаючого значення - рядок (в подвійних лапках), ціле число, з плаваючою точкою або логічне значення.

Робота з JSON в PHP

json_encode ($ data) - кодування даних у формат JSON json_decode ($ data) - декодування даних з JSON json_last_error ($ data) - повернути останні повідомлення про помилку

Технологія JSONP JSONP (JSON Padding) або «JSON з підкладкою» є розширенням JSON,

коли ім'я функції зворотного виклику вказується в якості вхідного аргументу.

Відмітна особливість JSONP в тому, що вона дозволяє звернутися до віддаленого серверу, передавши в якості додаткового GET-параметра callback-функцію, яку і потрібно викликати на сервері.

Реалізація JSONP-сервера

Сервер повинен повертати JSON, наприклад, таким чином:

```
header('Content-Type: text/html; charset=utf-8');
$callback = $_GET['callback'];
$param = $_GET['request'];
echo "$callback({param: \"$callback\", ip:
\"{$_SERVER['REMOTE_ADDR']}\"});";
```

Тоді отримати дані від сервера можна наступним чином (JQuery):

```
var surl = baseurl + $('#search').val() + '&callback=?'
$.getJSON(surl, function(data)
{
$("#result").html('request param: ' + data.param + '<br />ip: ' +
data.ip);
});
```

Використання JSON виправдано, якщо приймаюча сторона для обробки даних використовує JavaScript. Тобто дані повертаються у вигляді об'єкта JavaScript і не потрібно проводити парсинг.

Розглянемо приклад (chat.js)

```
function chatResult2 (msgs) {

// Додавання нових повідомлень в масив.

    for(var i = 0; i < msgs.length; i++)
      {
            var m = new Object();
            m.id = msgs[i]['id'];
            m.dt = msgs[i]['dt'];
            m.name = msgs[i]['name'];
            m.text = msgs[i]['text'];
            _messages.push(m);
// Додавання елементів в кінець масиву _messages

            _maxId = m.id;

}

// Виведення масиву повідомлень.

    var html = '';
      for (var i = _messages.length - 1; i >=0; i--)      {
            var m = _messages[i];
            html += '<b>' + m.name + '</b> ';
            html += m.dt;
            html += '<br/>';
            html += m.text;
            html += '<hr/>';
      }
    }
```

Тут з масиву асоційованих масивів msgs формується об'єкт m, з якого формується рядок повідомлення html.

Розглянемо серверний скрипт (ajax_json.php).

```
$maxId = $_POST['maxId'];
$messages = msg_select();
$messages = array_reverse($messages);
// Повертає масив у зворотному порядку

for($i = 0; $i < count($messages); $i++){
    if($messages[$i]['id_msg'] < $maxId){
        unset($messages[$i]);
// Видалення старих повідомлень з масиву, хоча простіше це було б зробити запитом до MySQL

continue;
    }
    $messages[$i]['name']= iconv('CP1251', 'UTF-8',
$messages[$i]['name']);
    $messages[$i]['text'] = iconv('CP1251', 'UTF-8',
$messages[$i]['text']);
// Iconv - перетворення кодувань з CP1251 в UTF-8, так як JSON використовує тільки UTF-8.

}

echo json_encode ($ messages);

// Повертає JSON-представлення даних
```

Робота з JSON в PHP

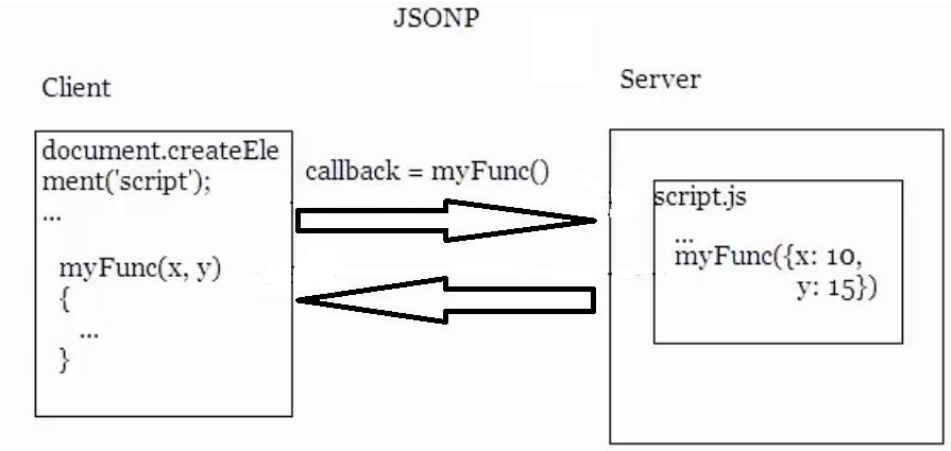
json_encode ($ data) - кодування даних у формат JSON

json_decode ($ data) - декодування даних з JSON

json_last_error ($ data) - повернути останні повідомлення про помилку

Технологія JSONP

JSONP (JSON Padding) або «JSON з підкладкою» є розширенням JSON, коли ім'я функції зворотного виклику вказується в якості вхідного аргументу. Необхідність цієї технології виникла через обмеження безпеки, що забороняє AJAX завантажувати дані з декількох доменів (тільки з одного, звідки була завантажена сторінка). Відмітна особливість JSONP в тому, що вона дозволяє звернутися до віддаленого серверу, передавши в якості додаткового GET-параметра callback-функцію, яку і потрібно викликати на сервері. Наявність в запиті записи 'callback =?' Говорить про те, що використовується JSONP.



Реалізація JSONP-сервера

Сервер повинен повертати JSON, наприклад, таким чином:

```
header('Content-Type: text/html; charset=utf-8');
$callback = $_GET['callback'];
$param = $_GET['request'];
echo "$callback({param: \"$callback\", ip:
\"{$_SERVER['REMOTE_ADDR']}\"});";
```

Тоді отримати дані від сервера можна наступним чином (JQuery):

```
var surl = baseurl + $('#search').val() + '&callback=?'
$.getJSON(surl, function(data){
$("#result").html('request param: ' + data.param + '<br />ip: ' +
data.ip);
});
```

Розглянемо приклад (json_ex.html)

```
function sendJSONP(url, callback)
{
  if (!url || !callback)
        return;
// Функція приймає два параметри і конструює з них один рядок

  url + = '& callback =' + callback;

// В атрибут callback записується назва функції

// Виконуємо запит JSONP

if (oElem) {// робота з елементом DOM, в якому може бути виклик скрипта, тому його очищаємо перед заповненням

oElem.parentNode.removeChild(oElem);
  }
  oElem = document.createElement('script');
  oElem.setAttribute('type','text/javascript');
  document.getElementsByTagName('head')[0].appendChild(oElem);
// Додавання цього елемента в head
```

oElem.setAttribute ('src', url); // встановлюємо створений атрибут із зверненням до зовнішнього скрипту, який поверне виклик встановленої функції callback

}

Як функції callback використовується

function loadit (data)

{

// Працюємо з даними

if (! data) return;

var oTotal = document.getElementById ("total");

oTotal.innerHTML = data.ResultSet.totalResultsAvailable;

}

Функція приймає дані data і виводимо як елемент html і проводимо її виклик

sendJSONP('http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=YahooDemo&query='+encodeURIComponent(query)+'&output=json', 'Loadit');

Приклад серверної частини (jsonp_server.php)

header ('Content-Type: text/html; charset = utf-8');

$ Callback = $ _GET ['callback']; // обов'язкове значення

$ Param = $ _GET ['request'];

echo "$ callback ({param: \" $ callback \ ", // параметри

ip: \"{$_SERVER['REMOTE_ADDR']} \"});"; //IP-адреса визиваючої сторони

Звернення до цього скрипту має вигляд (json_jquery_ex2.html)

```
var baseurl = 'http://localhost/WWW/jsonp_server.php?request='
      function search(){
         var surl = baseurl + $('#search').val() + '&callback=?'
         $.getJSON(surl, function(data)    {
      $("#result").html('request param:'+data.param+'<br />ip:'+ data.ip);
         });
      }
      $(document).ready(function(){
         $("#searchBtn").click(search)
      });
      </script>
```

data являє собою об'єкт з двома полями - параметр і ip. Результат роботи

Отримали значення параметра (назва функції callback, встановлене jQuery) та адресу сервера, з якого був зроблений запит.

Контрольні питання.

1. В чому відміна JSON від XML?
2. В чому переваги JSON від XML?
3. В чому недоліки JSON від XML?
4. З якою кодовою сторінкою працює JSON?

Частина 3.JavaScript

Тема 1:Використання JavaScript для розробки динамічних web-сторінок

JavaScript має наступні властивості

- Нестрога типізація
- Прототипно-орієнтований
- Сценарний (scripting)
- Найчастіше використовується у браузерах

JavaScript має деякі тонкощі. По перш всього, зазначимо, що з C прийшло те, що кожен оператор закінчується крапкою з комою. Навіть, якщо цей символ відсутній, то JavaScript поставить його автоматично!

```
function foo() {
    return  //Тут автоматично підставиться крапка з комою (у кінці строки)!
    {
        foo: ‘bar’
    }
}
function foo() {
    return {
        foo: ‘bar’
    }
}
```

Інший приклад – неочевидні факти роботи с масивами

```
1== 1; //true   ‘foo’== ‘foo’; //true   [1,2,3]== [1,2,3]; //false
```

Це витікає з того, що масив - це покажчик, а різним масивам відповідають різні покажчики.

```
new Array(3)== “,,”; //true
```

Пустий масив дорівнює рядку “,,”, так як у цьому випадку масив конвертується в рядок

```
new Array(3).toString(); //”,,”
```

але абсолютне порівняння дає

```
new Array(3)=== “,,”; //false
```

Робота з числами

Тільки один тип (float64, 8 байт с плаваючою комою)

0.1+0.2 == 0.3; // false 0.3000000… 1 == 1.0//true

1/0 = Infinity; -1/0 = -Infinity

NaN = не числове значення

1. Будь яка операція з NaN дає NaN

2. NaN != NaN

3. isNaN(...) Перевірка на NaN.

Деякі корисні функції

Number(10); // 10 Number(“42.23”); // 42.23 Number(“71oshi”); // NaN

parseInt(“18”); // 18 parseInt(“19kdjas”); // 19 parseInt(“74.54”); // 74

parseFloat(“74.54”); // 74.54

parseInt(num, base) за умовчанням – база=10, але якщо 0x… то база=16.

parseInt(“ff”); //NaN parseInt(“ff”,”16”); //255 parseInt(“0x10”); // 16
parseInt(“0x10”,”10”); // 0

parseFloat(num)

Перетворення

var x = 123456789; // undefined

x.toExponential(); //’1.23456789e+8’ x.toExponential(1); //’1.2e+8’

x.toExponential(2); //’1.23e+8’ x.toExponential(3); //’1.235e+8’

var y = 43.81327; //Обрізка числа

y.toFixed(); //’44’ y.toFixed(1); //’43.8’ y.toFixed(2); //’43.81’ y.toFixed(3);
//’43.813’

Текстові рядки

var string1 = “本語”;

var string2 = ‘terrible’; string2.length; // 8

“dance”.length; // 5 ‘42’.length; // 2

var uni = “\u1552”; // “□” “\u1552”.length; // 1 Юнікод

Екранування заборонених символів

‘it’s my life’; ‘it\’s my life’; ‘\u1552’;

Текстові рядки vs. символи

В js нема типу даних для символу. Символ це рядок с одним символом.

“abcdef”.charAt(2); // “c” “abcdef”.charAt(200); // ““ “abcdef”.charAt(-1); // ““

Конкатенація

“abcdef”.charAt(0) + “abcdef”.charAt(2) + “abcdef”.charAt(4); // “ace”

“together” + “ again”; // “together again”

12 + “ or “ + “20”; // “12 or 20”

””+1 //’1’ Пустий рядок+ щось=строка

””+null //’null’

Строки і числа

12 + “ or “ + “20”;// “12 or 20”

“12” / 2 + 1; //7

“day” * 2; // NaN

var a = 42; // 42 a.toString(); // “42”

a.toString().length; // 2

a.toString().length.toString(); // “2”

“Blink “ + 182; // “Blink 182”

“Blink “ + 181 + 1; // “Blink 1811”

“Blink “ + (181 + 1); // “Blink 182”

“1”+0 //’10’ но

+"1"+ +"0" //1 //унарний плюс переводить в число.

+"0" //0

+false //0

+null //0

+true //1 тобто, true>false

+undefined //NaN

Об’єкт в JS це не те ж саме, що об’єкт в класичних мовах ООП (екземпляр, реалізація класу)

Об’єкт– контейнер з властивостями (схоже на структури С). Все– об’єкти(окрім чисел, строк, true/ false-змінних, null та undefined). Функція– об’єкт, масив– об’єкт.

Ясно, що

var i=5;

var a=i; //5

i=10;

```
a                //5
але
var obj={count:5}
var obj1=obj;          //Зберігається посилання
obj.count=10;
obj1.count             //10
```

Об’єкт

```
var person = {
        “name” : “Alex”,
        “age”  : 25
        ““     : “Hello!”
};
person.name;      person [“name”];      // “Alex”
person.age;   person [“age”];    // 25
person [““];
//Hello! Доступу через крапку до пустого елементу нема
```

Вкладеність

```
var person = {
        name : “Alex”,
        wife : {
                name : “Eve”,
                age : 20
                },
        age : 25
};
        person [“name”];  // “Alex”
        person [“age”];  // 25
        person.wife;  // Object
        person.wife.age;  // 20
        person.wife.wife;  // undefined
```

Таким чином, функція-об'єкт, з властивостями name, length і prototype. Може використовуватися як будь-який інший об'єкт: зберігатися в змінних, інших об'єктах, передаватися як аргумент і повертати як значення.
Функцій можна задавати властивості і методи (!)
На відміну від інших об'єктів, функцію можна викликати, причому, якщо функція має вигляд
function f (...) {...} // function declaration
то вона генерується при завантаженні сторінки і може бути викликана в будь-якому місці коду, а
var f = function (...) {...} // function-expression
запускається під час її виклику.
Головна функція JavaScript – робота з DOM (Document Object Model). DOM - це не залежний від платформи і мови програмний інтерфейс, що дозволяє програмам і скриптам отримати доступ до вмісту HTML, XHTML і XML-документів, а також змінювати вміст, структуру та оформлення таких документів.
Приклад.

```
<!doctype html>
<html>
        <head>
        </head>
        <body>
        <input type="button" value="hide" id="hide"/>
        <input type="button" value="view" id="view"/>
        <br/>
        <img src='image/ddtu.jpg' id="info">
        </body>
           <script type="text/javascript">
              function addHideHandler(sourceId, targetId, param) {
        // Створений об'єкт [[scope]] з властивостями sourceId, targetId
        // Записати в [[scope]] властивість sourceNode
                   var sourceNode = document.getElementById (sourceId)
        // Записати в [[scope]] властивість handler
                         var handler = function() {
                                 var targetNode = document.getElementById(targetId)
                     targetNode.style.display = param;
                   }
                   sourceNode.onclick = handler
        // функція закінчила виконання
              }
              addHideHandler('hide', 'info','none');
              addHideHandler('view', 'info','block');
        /*
        Зовнішня функція закінчила свою роботу, але внутрішня - може запуститися коли-небудь потім.
        Інтерпретатор javascript не проводить аналіз - чи знадобляться внутрішньої функції змінні з зовнішньої, і які змінні можуть бути потрібні.
        Замість цього він просто залишає весь [[scope]] зовнішньої функції. Якщо після запуску внутрішня функція раптом не знайде яку-небудь змінну у своєму [[scope]] - вона може звернутися до [[scope]] зовнішньої функції і знайти там.
```

```
        Якщо зовнішня функція була створена всередині ще однієї (ще більш
        зовнішньої) функції - то в ланцюжок додається ще один [[scope]] і так - до
        глобальної області window.
        Зауважимо, що якщо перемістити функцію в заголовок, то вона створиться
        до того, як з'являться елементи з id   */
         </script>
</html>
```

Контрольні питання.

1. Яка відміна між BOM та DOM?
2. З якого елементу починається обхід дерева DOM?
3. Що представляє собою об’єкт window?
4. Як проводиться реєстрація подій?
5. Як працює скрипт «акордеон»?
6. Що є замикання?
7. Яка інформація заноситься в [[scope]]?

Тема 2: Асинхронний зв’язок з сервером. Підвантаження даних з серверу.

Asynchronous JavaScript and XML («асинхронний JavaScript і XML») - підхід до побудови інтерактивних інтерфейсів веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером.
В основі AJAX лежить Microsoft Remote Scripting (1998). Ідея була в створенні об'єкта MSXML, в який впроваджувався об'єкт XMLHTTP, що дозволяв зробити запит на сервер без перезавантаження сторінки.
В основі AJAX лежить об'єкт XmlHttpRequest (XMLHTTP)

Свойства	Методы
• onreadystatechange	• abort()
• readyState	• getAllResponseHeaders()
• responseText	• open()
• responseXML	• send()
• status	• setRequestHeader()
• statusText	

Створення об'єкта XmlHttpRequest

```
function getXmlHttpRequest(){
        if(window. XMLHttpRequest){
                try{ //Сучасні браузери
                        return new XMLHttpRequest();
                }catch(e){}}
```

```
else if (window.ActiveXObject){
        try{//IE6
            return new ActiveXObject('Msxml2.XMLHTTP');
        }catch(e){}
        else
            try{//IE5
                return new ActiveXObject('Microsoft.XMLHTTP');
            }
                catch (e){}}
            return null;
}
```

Синхронний запит очікує завершення звернення до сервера. Асинхронні запити виконуються паралельно в іншими операціями. Для контролю стану використовується властивість readyState і подія onreadystatechange.

Виконання синхронного запиту

```
// Об'єкт XMLHttpRequest
var request = get XMLHttpRequest ();
// Запит серверу
request.open ("GET", url, false);
// False- вказує на використання синхронного запиту
request.send (null);
// відіслали і чекаємо (всі підвисає в очікуванні відповіді)
```

Виконання асинхронного запиту

```
var request = get XMLHttpRequest ();
// Установка обробника
request. onreadystatechange = function () {
if (req.readyState == 4)
/* Значення стану
        0-нічого не сталося,
        1 з'єднання встановлено,
        2-запит відправлено,
        3-очікування відповіді,
        4 відповідь отримана і розібрана.
*/
...
}
// Запит на сервер
request.open ("GET", url, true);
request.send (null);
```

При використанні синхронного з'єднання кнопки не реагують на натискання, на відміну від асинхронного.

Отримання даних з сервера

• Властивість responseText-текст, отриманий від сервера.
• Властивість responseXML-XML DOM, отриманий від сервера.
• Властивість status- HTTP статус сервера.
• Метод getAllResponseHeaders () - заголовки відповідей з сервера.

Приклад.

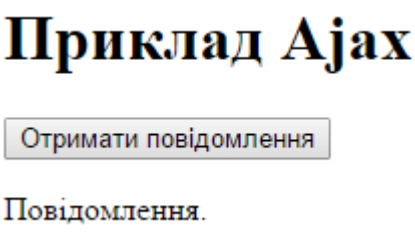
Файл ajax.html

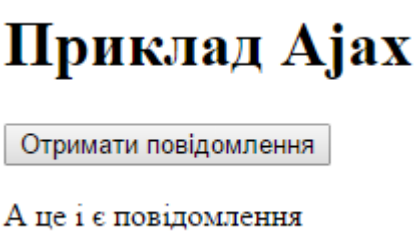
```
<html>
<head>
<title>Приклад Ajax</title>
<script language = "javascript">
var XMLHttpRequestObject = false;
if (window.XMLHttpRequest) {
        XMLHttpRequestObject = new XMLHttpRequest();
} else if (window.ActiveXObject) {
        XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
}
function getData(dataSource, divID)
{
if(XMLHttpRequestObject) {
        var obj = document.getElementById(divID);
        XMLHttpRequestObject.open("GET", dataSource);
        XMLHttpRequestObject.onreadystatechange = function()
{
if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
        obj.innerHTML = XMLHttpRequestObject.responseText;
}
}
XMLHttpRequestObject.send(null);
}
}
</script>
</head>
<body>
<H1>Приклад Ajax </H1>
<form>
<input type = "button" value = "Отримати повідомлення"
onclick = "getData('data.txt', 'targetDiv')">
</form>
<div id="targetDiv">
<p>Повідомлення.</p>
</div>
</body>
</html>
```

Файл data.txt

А це і є повідомлення

Результат роботи





Контрольні питання.

1. Яка відміна синхронним та асинхронним запитом?
2. Як вказати кодову сторінку при роботі з AJAX?
3. Чи можна використовувати формат JSON при роботі з AJAX?
4. Як реалізувати завантаження скрипта JavaScript з серверу?
5. Чи обов'язковий рядок XMLHttpRequestObject.readyState == 4 при роботі з AJAX?

МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ

1. Шумейко А.А. Web-программирование http://178.219.93.18:8080/Portal/Data/3/19/NeT/index.html

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова

2. Хокинс С. Администрирование Web-сервера Apache и руководство по электронной коммерции. Вильямс, 2001, 336 с.
3. Фленов М. Web-сервер глазами хакера. BHV-СПб, 2009, 320 с.
4. Гарнаев А., Гарнаев С. Web-программирование на Java и JavaScript-СПб.:bhv, 2002, 1040 с.
5. Зандстра Мэт. PHP4. Издательский дом "Вильямс", 2001, 374 с.
6. Нильсен Якоб. Веб-дизайн. Символ-плюс, СПб, 2002, 512 с.
7. Спортак Марк Ф. и др. Компьютерные сети. "ДиаСофт", 1999, 430 с.
8. Томсон Лаура, Веллинг Люк. Разработка Web-приложений на PHP и MySQL. "ДиаСофт", 2002, 655 с.
9. Шарма Вивск, Шарма Раджив. Разработка Web-серверов для электронной коммерции. Комплексный подход.Издательский дом "Вильямс", 2001, 397 с.
10. Вагнер Ричард, Вайк Аллен. JavaScript. Энциклопедия пользователя. ДиаСофт, 2001, 464 с.

Допоміжна

11. Келли Мэрдок. JavaScript. Наглядный курс создания динамических Web-страниц. Диалектика, 2005, 288 с.
12. Дейв Крейн, Эрик Паскарелло, Даррен Джеймс Ajax в действии. Диалектика, 2007, 640 с.
13. Д. В. Николенко Практические занятия по JavaScript. Наука и техника, 2000, 128 с.

ІНФОРМАЦІЙНІ РЕСУРСИ

14. http://www.dstu.dp.ua
15. http://www.codecademy.com/
16. https://www.codeschool.com/

НАВЧАЛЬНЕ ВИДАННЯ

Конспект лекцій з дисципліни «Технології створення Web-застосувань» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інженерія програмного забезпечення» зі спеціальності 121 – «Інженерія програмного забезпечення» галузі знань 12 – «Інформаційні технології»

Укладач:

проф. Шумейко Олександр Олексійович

Підписано до друку_____ 2019р.
Формат __А4__
Обсяг__8,3__ др. арк.
Наклад _____ прим. Замовлення
51918, м. Кам’янське,
вул. Дніпробудівська, 2