E4X

(ECMAScript 4 XML)

Autor: Piotr Kuśka

Plan prezentacji

1. Wprowadzenie
2. Składnia
3. Przykład zastosowania – Web services
4. Podsumowanie

1. Wprowadzenie

Co to jest ECMAScript?

- ECMAScript to nazwa standardu dla JavaScriptu
- Zatwierdzony przez European Computer Manufacturers Association

ECMAScript for XML (E4X)

- E4X jest rozszerzeniem ECMAScriptu o obsługę XML-a
- Standard jest alternatywą dla interfejsów DOM, oferującą prostą składnię do manipulacji dokumentami XML
- Pozwala na zanurzanie kodu XML w ECMAScripcie

Implementacje E4X

- Pierwsza wersja standardu E4X została opublikowana w czerwcu 2004 roku (ECMA-357), druga edycja w grudniu 2005.
- SpiderMonkey (przeglądarki Mozilli)
- Rhino (darmowa biblioteka JavaScript dla Javy)

2. Składnia

Tworzenie obiektów XML w JavaScripcie

- Nowy typ XML jako zwykły obiekt

```
var aDate = new Date();
var xmlObj = new XML (”<person><name>Ala</name></person>”);
```

- Można to napisać prościej

```
var xmlObj2 =
        <person id=”10”>
                <name>Ala</name>
                <role>ma kota</role>
        </person>;
```

Dostęp do obiektów XML

- Operator .

```
print(xmlObj.name);
print(”Osoba numer ” + xmlObj2.@id + ” – to ” +
        xmlObj2.name);
```

- @ - odwołanie do atrybutu

Obsługa list

```
var company =
      <company>
        <employee id=”1”><name>Jan Kowalski</name></employee>
        <employee id=”2”><name>Krzysztof Nowak</name></employee>
      </company>;
```

- Co zwróci `company.employee`?

Listy – c. d. (pętla `for each`)

```
print(company.employee[1].name);

for each (var emp in company.employee) {
  print(emp.name);
}

//uwaga, company.employee.name to lista imion wszystkich pracowników

for (var i in company.employee.name) {
  print(company.employee.name[i]);
}
```

Do list można dodawać elementy

```
company.employee +=
     <employee id=”3”><name>Edmund</name></employee>;
```

- Listy łatwo tworzy się przy użyciu operatora +

```
var newEmpList = <employee><name>Czesio</name></employee> +
                 new XML(”<employee><name>Zbysio</name></employee>”);
```

Operator ..

- Działa podobnie jak operator ., ale przeszukuje wszystkich potomków

```
for each (var name in company..name) {
  print(name);
}
```

Wybieranie tagów spełniających predykat

- Można wyszukiwać obiekty według kryterów:

```
var emp = company.employee.(@id == 1);
print (emp.name);
```

- Predykaty stosuje się także dla operatora ..

Modyfikacja zawartości

- Aby zmienić zawartość obiektu xml można posłużyć się operatorem .

```
emp.name = ”Janek”;
```

- Uwaga – zmienna `emp` jest referencją

```
print(company.employee[0].name);
```

Dodawanie nowych tagów

```
emp.height = ”180”;
company.employee[1].height = ”190”;
company.employee[1].job=”pilot”;
```

- … oraz atrybutów:

```
emp.height.@measure = ”cm”;
```

Obiekty można także usuwać

- Do tego służy operator `delete`

```
delete company.employee[1].job;
delete emp.height.@measure;
emp.height = new Number(emp.height) / 100;
emp.height.@measure = ”m”;
```

Poruszanie się po hierarchii obiektów

- Można odwołać się do rodzica obiektu za pomocą funkcji `parent()`

```
print(emp.parent());
```

- … oraz do dzieci

```
print(company.children().(name == ”Janek”).height);
```

Wyrażenia JavaScript w XML-u

- Aby wykonać wyrażenia JavaScriptowe w XML-u należy użyć nawiasów {}

```
var xmlName = <name>{company..name[0].toUpperCase()}</name>;
var tagname = ”item”;
var attr_name = ”id”;
var attr_value = ”9”;
var content = ”simple item”;
var x = <{tagname} {attr_name}={attr_value}>{content}</{tagname}>;
```

E4X obsługuje też wildcardy

- Zarówno dla tagów…

```
for each(var tag in company.employee[1].*) {
   print(tag);
}
```

- … jak i atrybutów

```
var myTag = <tag attr1=”1” attr2=”2” attr3=”3”/>
print(myTag.@*[2]);
```

Funkcje wbudowane w E4X

- `name()`- zwraca nazwę taga

```
print(emp.name());
```

- `contains(value)`- sprawdza, czy dany obiekt XML-owy zawiera value

```
company..name.contains(”Rysiu”) //false
company..name.contains(”Janek”); //true
```

- `length()` – zwraca długość listy

```
print(company.length());
print(company.employee.length());
```

Funkcje – c. d.

- `toString()` – przekształca element na napis

```
var x = <x id="a">10</x>;
print(x.toString());
```

- `toXMLString()` – zwraca napis, z którego można utworzyć nowy obiekt XML

```
print(x.toXMLString());
```

Namespace'y

- Namespace'y można wstawiać bezpośrednio tak jak obiekty XML-owe

```
var soapMsg = <s:Envelope
                xmlns:s="http://www.w3.org/2003/05/soap-envelope"/>;
print(soapMsg.namespace());
```

- Można ustawić domyślny namespace przed utworzeniem obiektu

```
default xml namespace =
  new Namespace("http://www.w3.org/2003/05/soap-envelope");
...
default xml namespace = "";
```

Namespace'y – c. d.

- Można je także wstawiać przy pomocy operatora ::

```
var f = new Namespace("http://fremantle.org");
soapMsg.Body.f::GetStockQuote="IBM";
print(soapMsg);
```

3. Przykład zastosowania – Web services

Co będzie nam potrzebne

- Aby udostępnić Web service potrzebny jest Axis uruchomiony na Tomcacie
- Klient korzystający z Web service'u za pomocą E4X będzie potrzebował przeglądarki obsługującej ten standard (np. Mozilla)

Prezentacja działania

- Zobaczmy, jak to działa w praktyce

4. Podsumowanie

Zalety E4X

- E4X ułatwia korzystanie z XML-a – nie trzeba korzystać z żadnych zewnętrzych bibliotek
- Pozwala na pisanie krótkiego kodu
- Jest łatwy do opanowania, bo rozszerza dobrze znany standard JavaScript

Wady

- Poważną wadą jest brak obsługi E4X przez Internet Explorer 7. To rozszerzenie jest jednak coraz częściej używane, więc można się spodziewać, że kolejne wersje przeglądarki Microsoftu będą je obsługiwały.

Dziękuję za uwagę

- Pytania?