Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Лабораторная работа №4

«XML, работа с SAX и DOM парсерами»

Выполнил: Пристром Р.М.

Студент группы № 814301

Проверил: Лыщик А.П.

Минск 2020

Задание:

* определить схему XML-файла;
* определить структуры данных для представления в памяти содержимого      XML-файла после выполнения разбора;
* способ разбора с создание документа;
* реализовать программу на языке программирования Java;

Наш XML-файл представляет файл с ФИО работников, их емэйлов и зарплаты.

*<?*xml version="1.0"*?>*<Company xsi:noNamespaceSchemaLocation="schema.xsd" xmlns:xsi="schema.xsd">  
 <Employee>  
 <FirstName>Batura</FirstName>  
 <LastName>Artyom</LastName>  
 <ContactNo>296910067</ContactNo>  
 <Email>artyombatura@gmail.com</Email>  
 <Salary>1000</Salary>  
 </Employee>  
 <Employee>  
 <FirstName>Konstantin</FirstName>  
 <LastName>Petrykevich</LastName>  
 <ContactNo>291544631</ContactNo>  
 <Email>petrykevich@mail.ru</Email>  
 <Salary>1500</Salary>  
 </Employee>  
 <Employee>  
 <FirstName>Belov</FirstName>  
 <LastName>Martin</LastName>  
 <ContactNo>290005501</ContactNo>  
 <Email>martinbelow@gmail.com</Email>  
 <Salary>800</Salary>  
 </Employee>  
</Company>

Определение схемы XML, широко известное как XSD, является способом точного описания языка XML. XSD проверяет правильность структуры и словаря XML-документа по отношению к грамматическим правилам соответствующего языка XML.

Соответствующая схема данного XML-файла будет выглядеть так:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
 <xs:element name="Company">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="Employee">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element type="xs:string" name="FirstName"/>  
 <xs:element type="xs:string" name="LastName"/>  
 <xs:element type="xs:integer" name="ContactNo"/>  
 <xs:element type="xs:string" name="Email"/>  
 <xs:element type="xs:integer" name="Salary"/>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
</xs:schema>

Стоит отметить, что элемент *complexType* определяет сложный тип. Элемент сложного типа может содержать другие элементы и/или атрибуты. Элемент *sequence* определяет, что дочерние элементы должны появляться в последовательности. Каждый дочерний элемент может использоваться от 0 до бесконечного числа раз, но в данном случае указан параметр maxOccurs, который определяет, что нужно снять лимит на использование (ключевое слово "unbounded") .

Код программы(парсера)

package com.company;  
  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.DocumentBuilder;  
import org.w3c.dom.Document;  
import org.w3c.dom.NodeList;  
import org.w3c.dom.Node;  
import org.w3c.dom.Element;  
import java.io.File;  
import java.util.ArrayList;  
  
public class Main {  
  
 public static void main(String[] args) {  
 try {  
 File fXmlFile = new File("src/com/company/employee.xml");  
 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();  
 Document doc = dBuilder.parse(fXmlFile);  
  
 doc.getDocumentElement().normalize();  
 System.*out*.println("Root element :" + doc.getDocumentElement().getNodeName());  
 NodeList nList = doc.getElementsByTagName("Employee");  
  
 ArrayList<Employee> employees = new ArrayList<>();  
  
 for (int temp = 0; temp < nList.getLength(); temp++) {  
  
 Node nNode = nList.item(temp);  
  
 System.*out*.println("Current Element :" + nNode.getNodeName());  
  
 if (nNode.getNodeType() == Node.*ELEMENT\_NODE*) {  
  
 Element eElement = (Element) nNode;  
 employees.add(new Employee(  
 eElement.getElementsByTagName("FirstName").item(0).getTextContent(),  
 eElement.getElementsByTagName("LastName").item(0).getTextContent(),  
 Integer.*parseInt*(eElement.getElementsByTagName("ContactNo").item(0).getTextContent()),  
 eElement.getElementsByTagName("Email").item(0).getTextContent(),  
 Integer.*parseInt*(eElement.getElementsByTagName("Salary").item(0).getTextContent())));  
 }  
 }  
 for (Employee stuff : employees) {  
 System.*out*.println(stuff.toString());  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Определение структуры хранения данных:

package com.company;  
  
public class Employee {  
 String firstName;  
 String lastName;  
 Integer contactNumber;  
 String email;  
 Integer salary;  
  
 Employee(String firstName, String lastName, Integer contactNumber, String email, Integer salary) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.contactNumber = contactNumber;  
 this.email = email;  
 this.salary = salary;  
 }  
  
 @Override  
 public String toString() {  
 return "Employee{" +  
 "firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", contactNumber=" + contactNumber +  
 ", email='" + email + '\'' +  
 ", salary=" + salary +  
 '}';  
 }  
}

Данные класс включает в себя все требуемые поля и конструктор.

Из Documentbuilderfactory мы получаем Documentbuilder. Documentbuilder содержит API для получения экземпляров DOM Document из XML-документа.

File fXmlFile = new File("src/com/company/employee.xml");  
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();  
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Получаем корневой элемент документа

System.*out*.println("Root element :" + doc.getDocumentElement().getNodeName());

Получаем Nodelist, который содержит элементы будущей структуры staff в документе при помощи getElementsByTagName()

NodeList nList = doc.getElementsByTagName("Employee");

Далее проходимся циклом по nList и дальше парсим отдельно каждую ячейку. Соответственно, получаем нужные нам атрибут getAttribute(), а элемент - getElementsByTagName(). Далее заносим все данные в ArrayList, при этом определяем новые объекты класса Employee.

Пример работы приложения:

