Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

Инструкция

по выполнению лабораторной работы №26-27
«Разработка, отладка и испытание программ создания и обработки XML-документов»

Минск 2024 г.

Лабораторная работа № 26-27

Тема работы: «Разработка, отладка и испытание программ создания и обработки XML-документов»

1 Цель работы

Сформировать умения и навыки эффективного поиска в тексте по заданному шаблону с помощью регулярных выражений; формирование умений и навыков редактирования, замены и удаления подстроки, создания итоговых отчетов по результатам работы с текстом.

2 Задание

Добавить изменения в программу из лабораторной работы№19: производить чтение и запись данных в файл.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

XML документ - DOM, SAXParser

Существуют две стратегии обработки XML документов: DOM (Document Object Model) и SAX (Simple API for XML). Основное их отличие связано с тем, что использование DOM позволяет читать и вносить изменения в существующий XML-документ, а также создавать новый. Стратегия использования SAX основывается на том, что содержимое XML-документа только анализируется. XML-текст может быть больших размеров: DOM должен весь документ «заглотить» и проанализировать, а SAX-парсер обрабатывает XML-документ последовательно и не требует дополнительной памяти.

Для работы с XML-файлами Java располагает достаточно большим набором инструментов, начиная от встроенных возможностей, которые предоставляет Core Java, и заканчивая большим набором разнообразного стороннего кода, оформленного в виде библиотек. Сначала рассмотрим использование DOM для чтения XML-файла и создания нового файла/документа. А в заключение будет приведено описание и применение SAX-парсера SAXParser.

XML документ представляет собой набор узлов (тегов). Каждый узел может иметь неограниченное количество дочерних узлов, которые, в свою очередь, также могут содержать потомков или не содержать их совсем. Таким образом, строится дерево объектов. DOM - это объектная модель документа, которая представляет собой это дерево в виде специальных объектов/узлов *org.w3c.dom.Node*. Каждый узел *Node* соответствует своему XML-тегу и содержит полную информацию о том, что это за тег, какие он имеет атрибуты, какие дочерние узлы содержит внутри себя и т.д. На самой

вершине этой иерархии находится org.w3c.dom.**Document**, который является корневым элементов дерева.

Чтение XML-файла

Получить объект *Document* XML-файла можно следующим образом :

```
DocumentBuilderFactory dbf;
DocumentBuilder      db ;
Document           doc;

dbf = DocumentBuilderFactory.newInstance();
db  = dbf.newDocumentBuilder();
doc = db.parse(new File("data.xml"));
```

Чтобы найти какой-либо узел в дереве можно использовать метод *getElementsByTagName*, который возвращает список всех элементов :

```
NodeList Document.getElementsByTagName(String name);
. . .
// Пример использования метода getElementsByTagName
NodeList nodeList = doc.getElementsByTagName("tagname");
```

Метод *getElementsByTagName* является case-sensitive, т.е. различает прописные и строчные символы.

В цикле можно просмотреть все дочерние узлы. С помощью метода *getAttributes* можно узнать атрибуты узла. Метод *getNodeType* позволяет проверить тип узла:

```
NodeList children = node.getChildNodes();
for (int i = 0; i < children.getLength(); i++) {
  // дочерний узел
  Node node = children.item(i);
  if (node.getNodeType() == Node.ELEMENT_NODE) {
    // атрибуты узла
    NamedNodeMap attributes = node.getAttributes();
    Node nameAttrib;
    nameAttrib = attributes.getNamedItem("attrib_name");
    System.out.println ("" + i + ". " +
                        nameAttrib.getNodeValue());
  }
}
```

Создание XML-файла

Для создания нового объекта **Document** используйте следующий код :

```
DocumentBuilderFactory dbf;
DocumentBuilder      db ;
Document           doc;

dbf = DocumentBuilderFactory.newInstance();
db  = dbf.newDocumentBuilder();
doc = db.newDocument();
```

Элемент Element объекта *Document* создается с использованием метода createElement. Для определения значения элемента следует использовать метод setTextContent. Для добавления элемента в узловую запись используйте метод appendChild (Node). Элемент может содержать атрибуты. Чтобы добавить к элементу атрибут следует использовать метод setAttribute. Если элемент уже содержит атрибут, то его значение изменится.

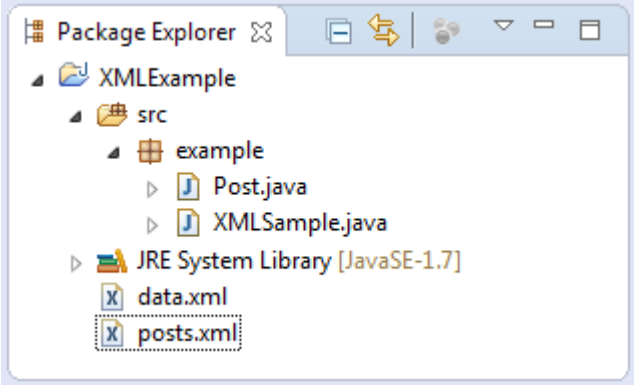
```
Element org.w3c.dom.Document.createElement(String s)
                        throws DOMException;
void org.w3c.dom.Node.setTextContent(String text)
                        throws DOMException;
Node org.w3c.dom.Node.appendChild(Node newChild)
                        throws DOMException;
void org.w3c.dom.Element.setAttribute(String name,
                       String value)
                        throws DOMException;

// Пример
Element root = doc.createElement("Users");

Element user = doc.createElement("user");
user.setTextContent("Остап Бендер");
user.setAttribute ("book", "12 стульев");
root.appendChild(user);
doc.appendChild(root);
```

В результате работы примера будет создан *Document* следующей структуры :

```
<?xml version="1.0" encoding="UTF-8"?>
<Users>
   <user book="12 стульев">Остап Бендер</user>
</Users>
```

Пример чтения и создания XML-файла

Для чтения готового XML-файла и формирования нового файла создадим в IDE Eclipse простой проект XMLSample, структура которого представлена на следующем скриншоте.



Проект включает XML-файл "posts.xml" с исходными данными, создаваемый XML-файл данных "data.xml", класс Post.java, в который будут

упаковываться отдельные записи массива данных и основной класс проекта XMLSample, который будет производить все необходимые действия.

Структура XML-файла

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="forum_name">...</field>
    <field name="year_post">...</field>
    <field name="post_subject">...</field>
    <field name="MID(post_text, 1, 80)">...</field>
    <field name="username">...</field>
    <field name="post_time">...</field>
    <field name="post_time">...</field>
    <field name="post_subject_source">...</field>
  </row>
</ROOT>
```

В качестве исходных данных используется XML-файл "posts.xml" из примеров разработчиков Sencha GXT 3.1.1. Структура XML-данных содержит корневой элемент <ROOT> и набор объектов/сущностей, представленных тегами <row />.

Листинг класса Person

```
import java.util.Date;

public class Post
{
  private static int ID = 0;

  private int    id;
  private String username;
  private String subject;
  private String forum;
  private Date   date;

  public Post() {
    setId(ID++);
  }
}
```

Класс Person имеет несколько полей. Идентификатор записи id определяется при создании объекта в конструкторе. Методы set/get не представлены в листинге.

Чтение XML-файла

Для чтения XML-файла в проекте используется метод readDataXML(), который создает список persons типа List<Person>, читает XML-файл данных и формирует объект doc типа *Document*. После этого в цикле создается массив данных. Вспомогательная функция getValue извлекает текст атрибута записи.

```
import org.w3c.dom.Node;
```

```
import org.w3c.dom.Element;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

. . .
public class XMLSample
{
  private  final  String     FILE_post = "posts.xml";
  private       List<Post>  posts;

  private String getValue(NodeList fields, int index)
  {
    NodeList list = fields.item(index).getChildNodes();
    if (list.getLength() > 0) {
      return list.item(0).getNodeValue();
    } else {
      return "";
    }
  }
  private void readDataXML()
  {
    posts = new ArrayList<Post>();

    SimpleDateFormat      sdf = null;
    DocumentBuilderFactory dbf = null;
    DocumentBuilder       db  = null;
    Document              doc = null;
    try {
      sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
      dbf = DocumentBuilderFactory.newInstance();
      db  = dbf.newDocumentBuilder();
      doc = null;

      FileInputStream fis = null;
      if (fileExists(FILE_post)) {
        try {
          fis = new FileInputStream(FILE_post);
          doc = db.parse(fis);
        } catch (FileNotFoundException e) {
          e.printStackTrace();
        }
      }
      doc.getDocumentElement().normalize();

      NodeList fields   = null;
      NodeList nodeList = null;
```

```
            nodeList = doc.getElementsByTagName("row");
            for (int s = 0; s < nodeList.getLength(); s++) {
              Node node = nodeList.item(s);
              if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element el = (Element) node;
                fields = el.getElementsByTagName("field");
                Post p = new Post();
                p.setForum (getValue(fields, 0));
                p.setDate  (sdf.parse(getValue(fields,1)));
                p.setSubject (getValue(fields, 2));
                p.setUsername(getValue(fields, 4));
                posts.add(p);
              }
            }
          } catch (Exception e) {
            e.printStackTrace();
          }
        }
      }
```

Следует обратить внимание, что для чтения значения атрибута записи (объекта Person) сначала получаем ссылку на массив тегов <field>, и после этого по индексу в функции getValue извлекаем значение.

Создание XML-файла

Для создания нового XML-файла на основе массива posts подготовим два списка данных типа List : пользователей users и форумов forums. Эти два массива запишем в XML-файл.

Создание нового объекта **Document** и сохранение его в XML-файл в проекте выполняет метод writeDataXML :

Листинг метода создания XML-файла

```
private final String FILE_data = "data.xml";

private void writeDataXML()
{
  DocumentBuilderFactory dbf = null;
  DocumentBuilder        db  = null;
  Document               doc = null;
  try {
    dbf = DocumentBuilderFactory.newInstance();
    db  = dbf.newDocumentBuilder();
    doc = db.newDocument();

    Element e_root  = doc.createElement("Posts");
    e_root.setAttribute("lang", "en");
    Element e_users  = doc.createElement("Users");
    Element e_forums = doc.createElement("Forums");
    e_root.appendChild(e_users);
    e_root.appendChild(e_forums);
    doc.appendChild(e_root);
```

```
            if (posts.size() == 0)
              return;

            List<String> users  = new ArrayList<String>();
            List<String> forums = new ArrayList<String>();
            for (int i = 0; i < posts.size(); i++) {
              if (!users.contains(posts.get(i).getUsername()))
                users.add(posts.get(i).getUsername());
              if (!forums.contains(posts.get(i).getForum()))
                forums.add(posts.get(i).getForum());
              }
               System.out.println("   пользователей : " +
                                  users.size());
            for (String user : users) {
              Element e = doc.createElement("user");
              e.setTextContent(user);
              e_users.appendChild (e);
            }
            System.out.println("   форумов : " + forums.size());
            for (String forum : forums) {
              Element e = doc.createElement("forum");
              e.setTextContent(forum);
              e_forums.appendChild (e);
            }
          } catch (ParserConfigurationException e) {
            e.printStackTrace();
          } finally {
            // Сохраняем Document в XML-файл
            if (doc != null)
              writeDocument(doc, FILE_data);
          }
        }
```

Процедура сохранения объекта **Document** в XML-файл представлена отдельным методом writeDocument :

Листинг процедуры сохранения XML-файла

```
        /**
         * Процедура сохранения DOM в файл
         */
        private void writeDocument(Document document, String path)
                  throws TransformerFactoryConfigurationError
        {
          Transformer     trf = null;
          DOMSource       src = null;
          FileOutputStream fos = null;
          try {
            trf = TransformerFactory.newInstance()
                              .newTransformer();
            src = new DOMSource(document);
```

```
            fos = new FileOutputStream(path);

                StreamResult result = new StreamResult(fos);
            trf.transform(src, result);
          } catch (TransformerException e) {
            e.printStackTrace(System.out);
          } catch (IOException e) {
            e.printStackTrace(System.out);
          }
        }
```

Если массив posts окажется пустым, то новый XML-файл должен будет иметь следующий вид :

```
<?xml version="1.0" encoding="UTF-8"?>
<Posts lang="en">
  <Users></Users>
  <Forums></Forums>
</Posts>
```

SAX-парсер, SAXParser

SAX-парсеры используют для анализа XML-строки или извлечения из нее необходимой информации. Обычно SAX-парсеры требуют фиксированный объем памяти и не позволяют изменять содержимое. Для связи SAX-парсера с вызывающим приложением, как правило, используется функция обратного вызова.

Рассмотрим пример SAXExample.java с использованием класса **SAXParser** для анализа XML-текста, представленного файлом phonebook.xml, содержащего 3 записи и имеющего следующий вид :

Файл phonebook.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phonebook>
  <person>
    <name>Остап Бендер</name>
    <email>ostap@12.com</email>
    <phone>999-987-6543</phone>
  </person>
  <person>
    <name>Киса Воробьянинов</name>
    <email>kisa@12.com</email>
    <phone>999-986-5432</phone>
  </person>
  <person>
    <name>Мадам Грицацуева</name>
    <email>madam@12.com</email>
    <phone>999-985-4321</phone>
  </person>
</phonebook>
```

Пакеты «javax.xml.parsers» и «org.xml.sax» включают набор классов для «разбора» XML в строковом представлении. К основным классам этих

пакетов, с точки зрения разложения XML объекта на составляющие, относятся *SAXParser* и *DefaultHandler*.

В примере SAXExample.java создается класс handler типа *DefaultHandler*, в котором методы анализа XML-строки переопределяются. Все прозрачно.

Листинг SAXExample.java

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXExample
{
  final String fileName = "phonebook.xml";
  final String TAG_NAME = "name";

  DefaultHandler handler = new DefaultHandler() {
    boolean tagOn = false; // флаг начала разбора тега

    /**
     * Метод вызывается, когда SAXParser начинает
     * обработку тэга
     */
    @Override
    public void startElement(String uri,
                    String localName,
                    String qName,
                    Attributes attributes)
                         throws SAXException {
      // Устанавливаем флаг для тега TAG_NAME
      tagOn = (qName.equalsIgnoreCase(TAG_NAME));
      System.out.println("\t<" + qName + ">");
    }

    /**
     * Метод вызывается, когда SAXParser считывает
     * текст между тэгами
     */
    @Override
    public void characters(char ch[],
                    int start, int length)
                         throws SAXException {
      // Проверка флага
      if (tagOn) {
        // Флаг установлен
        System.out.println("\t\t" +
                    new String(ch,start,length));
```

```
                tagOn = false;
            }
        }
        @Override
        public void endElement (String uri,
                    String localName,
                    String qName)
                        throws SAXException
        {
            super.endElement (uri, localName, qName);
        }

        @Override
        public void startDocument () throws SAXException
        {
            System.out.println ("Начало разбора документа!");
        }

        @Override
        public void endDocument () throws SAXException
        {
            System.out.println ("Разбор документа завершен!");
        }
    };

    public SAXExample ()
    {
        try {
            SAXParserFactory factory;
            factory = SAXParserFactory.newInstance ();
            SAXParser saxParser = factory.newSAXParser ();

            // Стартуем разбор XML-документа
            saxParser.parse (fileName, handler);

        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
    public static void main (String args[])
    {
        new SAXExample ();
        System.exit (0);
    }
}
```

5. Порядок выполнения работы

1. Выполнить задание в соответствии с вариантом индивидуального задания.

2. Отчёт дополнить описанием работы пользователя с разработанным программным средством.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Поясните две стратегии обработки XML документов.
2. Поясните назначение и использование пакетов «javax.xml.parsers» и «org.xml.sax».
3. Приведите пример создания XML-файла.
4. Каким образом возможно получить объект Document XML-файла?

8. Рекомендуемая литература

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

2. Прайс, М. Дж. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

3. Васильев, А.Н. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

4. Фримен, А. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.