нии по XML вызывает соответствующие методы у классов, реализующих интерфейсы, предоставляемые SAX-парсером.

- StAX (Streaming API for XML) не создает дерево объектов в памяти, но, в отличие от SAX-парсера, за переход от одной вершины XML к другой отвечает приложение, которое запускает разбор документа.

Анализаторы, которые строят древовидную модель, – это DOM-анализаторы. Анализаторы, которые генерируют события, – это SAX-анализаторы.

Анализаторы, которые ждут команды от приложения для перехода к следующему элементу XML – StAX-анализаторы.

В первом случае анализатор строит в памяти дерево объектов, соответствующее XML-документу. Далее вся работа ведется именно с этим деревом.

Во втором случае анализатор работает следующим образом: когда происходит анализ документа, анализатор вызывает методы, связанные с различными участками XML-файла, а программа, использующая анализатор, решает, как реагировать на тот или иной элемент XML-документа. Так, аннализатор будет генерировать событие о том, что он встретил начало документа либо его конец, начало элемента либо его конец, символьную информацию внутри элемента и т.д.

StAX работает как **`Iterator`**, который указывает на наличие элемента с помощью метода **`hasNext()`** и для перехода к следующей вершине использует метод **`next()`**.

Когда следует использовать DOM-, а когда – SAX, StAX -анализаторы?

DOM-анализаторы следует использовать тогда, когда нужно знать структуру документа и может понадобиться изменять эту структуру либо использовать информацию из XML-файла несколько раз.

SAX/StAX-анализаторы используются тогда, когда нужно извлечь информацию о нескольких элементах из XML-файла либо когда информация из документа нужна только один раз.

Событийная модель

Как уже отмечалось, SAX-анализатор не строит дерево элементов по содержимому XML-файла. Вместо этого анализатор читает файл и генерирует события, когда находит элементы, атрибуты или текст. На первый взгляд, такой подход менее естествен для приложения, использующего анализатор, так как он не строит дерево, а приложение само должно догадаться, какое дерево элементов описывается в XML-файле.

Однако нужно учитывать, для каких целей используются данные из XML-файла. Очевидно, что нет смысла строить дерево объектов, содержащее десятки тысячи элементов в памяти, если всё, что необходимо, – это просто посчитать точное количество элементов в файле.

SAX-анализаторы

SAX API определяет ряд методов, используемых при разборе документа:

`void startDocument()` – вызывается на старте обработки документа;

`void endDocument()` – вызывается при завершении разбора документа;

`void startElement(String uri, String localName, String qName, Attributes attrs)` – будет вызван, когда анализатор полностью обработает содержимое открывающего тега, включая его имя и все содержащиеся атрибуты;

void endElement(String uri, String localName, String qName) – сигнализирует о завершении элемента;

void characters(char[] ch, int start, int length) – вызывается в том случае, если анализатор встретил символьную информацию внутри элемента (тело тега);

warning(SAXParseException e), **error(SAXParseException e)**, **fatalError(SAXParseException e)** – вызываются в ответ на возникающие предупреждения и ошибки при разборе XML-документа.

В пакете **org.xml.sax в SAX2 API** содержатся интерфейсы **org.xml.sax.ContentHandler**, **org.xml.sax.ErrorHandler**, **org.xml.sax.DTDHandler**, и **org.xml.sax.EntityResolver**, которые необходимо реализовать для обработки соответствующего события.

Для того чтобы создать простейшее приложение, обрабатывающее XML-документ, достаточно сделать следующее:

1. Создать класс, который реализует один или несколько интерфейсов (**ContentHandler**, **ErrorHandler**, **DTDHandler**, **EntityResolver**) и реализовать методы, отвечающие за обработку интересующих событий.
2. Используя SAX2 API, поддерживаемое всеми SAX парсерами, создать **org.xml.sax.XMLReader**, например для Xerces:
   ```
   XMLReader reader =
   XMLReaderFactory.createXMLReader(
       "org.apache.xerces.parsers.SAXParser");
   ```
3. Передать в **XMLReader** объект класса, созданного на шаге 1 с помощью соответствующих методов:
   setContentHandler(), **setErrorHandler()**, **setDTDHandler()**, **setEntityResolver()** .
4. Вызвать метод **parse()**, которому в качестве параметров передать путь (URI) к анализируемому документу либо **InputSource**.

Следующий пример выводит на консоль содержимое XML-документа.

```
/* пример # 1 : чтение и вывод XML-документа : SimpleHandler.java */
package chapt16.analyzer.sax;
import org.xml.sax.ContentHandler;
import org.xml.sax.Attributes;

public class SimpleHandler implements ContentHandler {

   public void startElement(String uri, String localName,
         String qName, Attributes attrs) {
      String s = qName;
      //получение и вывод информации об атрибутах элемента
      for (int i = 0; i < attrs.getLength(); i++) {
         s += " " + attrs.getQName(i) + "="
            + attrs.getValue(i) + " ";
      }
      System.out.print(s.trim());
   }
```

```
   public void characters(char[] ch,
         int start, int length) {
      System.out.print(new String(ch, start, length));
   }
   public void endElement(String uri,
      String localName, String qName) {
      System.out.print(qName);
   }
}
```

/ пример # 2 : создание и запуск парсера : SAXSimple.java*/*

```
package chapt16.main;
import org.xml.sax.XMLReader;
import org.xml.sax.XMLReaderFactory;
import org.xml.sax.SAXException;
import javax.xml.parsers.ParserConfigurationException;
import java.io.IOException;
import chapt16.analyzer.sax.SimpleHandler;

public class SAXSimple {
   public static void main(String[] args) {
      try {
         //создание SAX-анализатора
XMLReader reader = XMLReaderFactory.createXMLReader();
        SimpleHandler contentHandler = new SimpleHandler();
        reader.setContentHandler(contentHandler);
        reader.parse("students.xml");
      } catch (SAXException e) {
         e.printStackTrace();
         System.out.print("ошибка SAX парсера");
      } catch (ParserConfigurationException e) {
         e.printStackTrace();
         System.out.print("ошибка конфигурации");
      } catch (IOException e) {
         e.printStackTrace();
         System.out.print("ошибка I/O потока");
      }
   }
}
```

В результате в консоль будет выведено (если убрать из XML-документа ссылку на DTD):

```
students
      student login=mit faculty=mmf
            name Mitar Alex name
            telephone 2456474 telephone
            address
                  country Belarus country
                  city Minsk city
                  street Kalinovsky 45 street
             address
```

```
       student
      student login=pus faculty=mmf
            name Pashkun Alex name
            telephone 3453789 telephone
            address
                  country Belarus country
                  city Brest city
                  street Knorina 56 street
             address
       student
students
```

В следующем приложении производятся разбор документа **students.xml** и инициализация на его основе коллекции объектов класса **Student**.

/ пример # 3 : формирование коллекции объектов на основе XML-документа : StudentHandler.java */*

```
package chapt16.analyzer.sax;

enum StudentEnum {
   NAME, TELEPHONE, STREET, CITY, COUNTRY
}
package chapt16.analyzer.sax;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import java.util.ArrayList;
import chapt16.entity.Student;

public class StundentHandler implements ContentHandler {
   ArrayList<Student> students = new ArrayList<Student>();
   Student curr = null;
   StudentEnum currentEnum = null;

   public ArrayList<Student> getStudents() {
      return students;
   }
   public void startDocument() {
      System.out.println("parsing started");
   }
   public void startElement(String uri, String localName,
         String qName, Attributes attrs) {
      if (qName.equals("student")) {
         curr = new Student();
         curr.setLogin(attrs.getValue(0));
         curr.setFaculty(attrs.getValue(1));
      }
      if(!"address".equals(qName) &&
         !"student".equals(qName) &&
         !qName.equals("students"))
         currentEnum =
            StudentEnum.valueOf(qName.toUpperCase());
   }
```

```
   public void endElement(String uri, String localName,
         String qName) {
      if(qName.equals("student"))
         students.add(curr);
      currentEnum = null;
   }
   public void characters(char[] ch, int start,
                                          int length) {
      String s = new String(ch, start, length).trim();
      if(currentEnum == null) return;
      switch (currentEnum) {
         case NAME:
            curr.setName(s);
            break;
         case TELEPHONE:
            curr.setTelephone(s);
            break;
         case STREET:
            curr.getAddress().setStreet(s);
            break;
         case CITY:
            curr.getAddress().setCity(s);
            break;
         case COUNTRY:
            curr.getAddress().setCountry(s);
            break;
      }
   }
}
```

/ пример # 4 : создание и запуск парсера : SAXStudentMain.java */*

```
package chapt16.main;
import org.xml.sax.XMLReader;
import org.xml.sax.XMLReaderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import java.util.ArrayList;
import chapt16.analyzer.sax.StundentHandler;
import chapt16.entity.Student;
import java.io.IOException;

public class SAXStudentMain {
   public static void main(String[] args) {
      try {
         //создание SAX-анализатора
        XMLReader reader =
            XMLReaderFactory.createXMLReader();
         StundentHandler sh = new StundentHandler();
        reader.setContentHandler(sh);
```

```
            ArrayList <Student> list;
            if(sh != null) {
               //разбор XML-документа
               parser.parse("students.xml");
                  System.out.println(sh.getStudents());
               }
         } catch (SAXException e) {
            e.printStackTrace();
            System.out.print("ошибка SAX парсера");
         } catch (ParserConfigurationException e) {
            e.printStackTrace();
            System.out.print("ошибка конфигурации");
         } catch (IOException e) {
            e.printStackTrace();
            System.out.print("ошибка I/O потока");
         }
      }
}
```

В результате на консоль будет выведена следующая информация:

```
parsing started
Login: mit
Name: Mitar Alex
Telephone: 2456474
Faculty: mmf
Address:
        Country: Belarus
        City: Minsk
        Street: Kalinovsky 45
Login: pus
Name: Pashkun Alex
Telephone: 3453789
Faculty: mmf
Address:
        Country: Belarus
        City: Brest
        Street: Knorina 56
```

Класс, объект которого формируется на основе информации из XML-документа, имеет следующий вид:

```
/* пример # 5 : класс java bean : Student.java */
package chapt16.entity;

public class Student {
   private String login;
   private String name;
   private String faculty;
   private String telephone;
   private Address address = new Address();
```

```
public String getLogin() {
   return login;
}
public void setLogin(String login) {
   this.login = login;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}
public String getFaculty() {
   return faculty;
}
public void setFaculty(String faculty) {
   this.faculty = faculty;
}
public String getTelephone() {
   return telephone;
}
public void setTelephone(String telephone) {
   this.telephone = telephone;
}
public Address getAddress() {
   return address;
}
public void setAddress(Address address) {
   this.address = address;
}
public String toString() {
   return "Login: " + login
      + "\nName: " + name
      + "\nTelephone: " + telephone
      + "\nFaculty: " + faculty
      + "\nAddress:"
      + "\n\tCountry: " + address.getCountry()
      + "\n\tCity: " + address.getCity()
      + "\n\tStreet: " + address.getStreet()
      + "\n";
}
public class Address {//внутренний класс
   private String country;
   private String city;
   private String street;

   public String getCountry() {
      return country;
   }
```

```
        public void setCountry(String country) {
            this.country = country;
        }
        public String getCity() {
            return city;
        }
        public void setCity(String city) {
            this.city = city;
        }
        public String getStreet() {
            return street;
        }
        public void setStreet(String street) {
            this.street = street;
        }
    }
}
```

Древовидная модель

Анализатор DOM представляет собой некоторый общий интерфейс для работы со структурой документа. При разработке DOM-анализаторов различными вендорами предполагалась возможность ковариантности кода.

DOM строит дерево, которое представляет содержимое XML-документа, и определяет набор классов, которые представляют каждый элемент в XML-документе (элементы, атрибуты, сущности, текст и т.д.).

В пакете **org.w3c.dom** можно найти интерфейсы, которые представляют вышеуказанные объекты. Реализацией этих интерфейсов занимаются разработчики анализаторов. Разработчики приложений, которые хотят использовать DOM-анализатор, имеют готовый набор методов для манипуляции деревом объектов и не зависят от конкретной реализации используемого анализатора.

Существуют различные общепризнанные DOM-анализаторы, которые в настоящий момент можно загрузить с указанных адресов:

Xerces – http://xerces.apache.org/xerces2-j/;

JAXP – входит в JDK.

Существуют также библиотеки, предлагающие свои структуры объектов XML с API для доступа к ним. Наиболее известные:

JDOM – http://www.jdom.org/dist/binary/jdom-1.0.zip.

dom4j – http://www.dom4j.org

Xerces

В стандартную конфигурацию Java входит набор пакетов для работы с XML. Но стандартная библиотека не всегда является самой простой в применении, поэтому часто в основе многих проектов, использующих XML, лежат библиотеки сторонних производителей. Одной из таких библиотек является Xerces, замечательной особенностью которого является использование части стандартных возможностей XML-библиотек JSDK с добавлением собственных классов и методов, упрощающих и облегчающих обработку документов XML.