Java course

Search

Go to ... Go to ...

XML — как это работает

Я поначалу совсем не хотел останавливаться на том, что такое XML, но все-таки несколько слов сказать придется, чтобы те, кто про него услышали первый раз, прочитав эти строки, были в курсе основных идей.
По сути XML представляет собой обычный текст, который разделяется на логические группы с помощью специальных меток, которые называют “тэг”.
Тэг представляет собой слово, которое заключено в угловые скобки — например вот так:

<test>

Для того, чтобы начать группу вы указывает просто слово в скобках — еще раз повторим **<test>**
Для окончание группы вы указываете такой же тэг, но слово предваряется символом “/”. Вот так: **</test>**

В итоге группа внутри тэга test выглядит вот так:
<test>Группа символов для тестирования</test>

Тэги могу вкладываться один в другой — например вот так:
<testGroup>
<test>Группа символов для тестирования 01</test>
<test>Группа символов для тестирования 02</test>

- Расширенное описание классов
- Исключения
- Решения на основе классов
- Список контактов — начало
- Коллекции — базовые принципы
- Коллекции — продолжение
- Список контактов — GUI приложение
- Что такое JAR-файлы
- Многопоточность — первые шаги
- Многопоточность и синхронизация
- Работаем с XML
- Reflection — основы
- Установка СУБД PostgreSQL
- Базы данных на Java — первые шаги
- Возможности JDBC — второй этап
- JDBC — групповые операции
- Список контактов — работаем с БД
- Переезжаем на Maven
- Потоки ввода-вывода
- Сетевое взаимодействие
- С чего начинается Web

<test>Группа символов для тестирования 03</test>
</testGroup>

Кроме вложений текста в тэгах можно указывать атрибуты — вот так:

<test attribute=”1” >Группа символов для тестирования 01</test>

Назначение тэгов очень простое — надо отметить/выделить/сгруппировать какую-то информацию для того, чтобы потом ее можно было использовать. Это может быть список имен, список книг, список фирм, список вакансий и т.д.
Например, я хочу написать список контактов, с указанием имени, фамилии и e-mail. Можно сделать это так (но можно и по-другому — здесь все зависит от вашей фантазии и требований задачи):

<contacts>
<contact type=”друг”>
<firstName>Василий</firstName>
<lastName>Курочкин</lastName>
<email>vas@pisem.net</email>
</contact>
<contact type=”коллега”>
<firstName>Георгий</firstName>
<lastName>Папанов</lastName>
<email>geogr@gmail.com</email>
</contact>
<contact type=”однокурсник”>
<firstName>Семен</firstName>
<lastName>Баринов</lastName>
<email>barinov@yandex.ru</email>
</contact>
</contacts>

Не ищите тайного смысла — я просто сделал строку, в которой выделил нужные мне части — контакт (тэг contact) и внутри определил имя, фамилию и e-mail (тэги firstName, lastName, email). Также с помощью атрибута type я определи тип контакта — друг, коллега, однокурсник. Теперь просматривая строку я могу выделить нужные мне части информации. Это удобно и ничего более. Причем здесь больше удобства даже не для визуального восприятия (это спорно), а для программной обработки — достаточно несложно написать программу, которая найдет конкретные кусочки.

Теперь новичкам надо посмотреть какой-нибудь XML, чтобы увидеть больше примеров и убедиться, что в главном я прав 🙂 (хотя все в мире относительно).

Работа с XML

В первую очередь я хотел бы высказать свою позицию по поводу самого XML и уже на основе этого продолжать повествование.
Для меня XML — очень мощная технология, которая позволяет хранить, передавать и обрабатывать сложноструктурированные данные. Т.е. если я хочу иметь: список фирм с их телефонами и счетами, каталог книг с авторами и отзывами, описание структуры страниц сайта с комментариями, состояние всех автобусов в городе с их координатами, водителями, номерами и прочая — все это может быть удобно сохранено в виде XML и, что крайне важно и удобно, может быть передано в любую систему, которая написана на любой платформе — на .NET, PHP, Object C, Delphi, C++.
Проведите мысленный эксперимент — попробуйте написать строку, в которой передать информацию о своих контактах (где у одной персоны может быть несколько телефонов, e-mail, любимые книги, места работы, места учебы и … да хватит пока). Что важно — это должна быть обычная строка (несколько строк), которая позволяет разбирать эту информацию в ту структуру, которую я описал — класс Java. Там надо предусмотреть какие-то разделители, информацию об именах полей (группах полей). Попробуйте — и вы придете к чему-то подобному XML.

Также я вам советую очень серьезно подойти к изучение самого XML, т.к. на сегодня эта технология используется в очень широком спектре всевозможных пакетов, библиотек, платформ и вам никуда от нее не скрыться.

Итак, после всего вышесказанного мы видим, что нам приходит строковое представление чего-то важного в какой-то определенной структуре, которая требует наличия достаточно важных функций — нам же надо как-то работать с этой информацией. Не лежать же ей мертвым грузом.
Функции достаточно очевидны:

1. Разбор. Надо уметь разобрать строку на что-то более удобное для обработки — пытаться вставить внутрь строки или находить какое-то поле определенной записи из строки — это достаточно неудобно. Значит нам надо иметь некоторый набор классов для представления нашей строки в виде структуры объектов.
2. Поиск. По структуре данных надо уметь что-то искать. Причем не подстроку, а какую-то группу полей, которые относятся к определенному объекту — например полная информация о книге — наименование, авторы, отзывы. Или список контактов с фамилией “Сидоров”.
3. Проверка. Данные должны быть корректными, т.е. там должны быть только определенные поля, с определенным наполнением и они должны быть правильно скомпанованы в нашей строке.
4. Преобразование. Хоть XML достаточно удобно описывает структурированные данные, это не значит, что его удобно просматривать обычному человеку или всегда удобно обрабатывать. Нередко для решения этого вопроса требуется преобразовать XML в какое-либо другое текстовое представление — например в тот же HTML (который является частным воплощением XML). Или даже в обычный текст.

В общем-то это все, что на мой взгляд наиболее важно. Нам надо уметь работать с информацией, которая записана в формате XML. Приступим к рассмотрению каждого пункта.

Разбор — Parsing

В слэнге программистов часто используется слова “парсинг”, который и обозначает разбор строки (или еще чего-нибудь) в какую-то структуру. Здесь надо выделить два момента:

1. Разбор строки определенным алгоритмом

2. Сохранение результатов разбора в какую-то структуру вместо строки — ибо со строкой многие операции делать просто неудобно.

Что касается второго пункта, то на сегодня существует по сути одна унифицированная структура, которая называется Document Object Model (DOM). DOM представляет собой набор интерфейсов (и их реализаций), которые являются специализированными объектами для хранения “узлов” (node) XML-документа. По сути каждый тэг — это “узел” (нода — я буду использовать этот термин, т.к. очень привык). Информация внутри тэга — тоже нода. По сути любой разобранный XML — это набор элементов типа Node и еще более специализированных, построенных в дерево.
Почему дерево ? Да потому что у каждой ноды может быть список дочерних нод и у каждой из них тоже может быть “детки”. Так и строится дерево.
Если вам сложно это увидеть, то советую посмотреть какую-либо классическую книгу по алгоритмам и структурам данных — можно старого доброго Никлауса Вирта «Алгоритмы и структуры данных». Книгу можно найти в интернете, ну или купить.

DOM

Давайте рассмотрим загрузку файла и редактирование построенного дерева на примере. Наша программа считывает XML-файл со списком книг и печатает их свойства. Сам XML-файл выглядит вот так:

```
<?xml version="1.0" encoding="UTF-8"?>
<BookCatalogue>
    <Book>
        <Title>Yogasana Vijnana: the Science of Yoga</Title>
        <Author>Dhirendra Brahmachari</Author>
        <Date>1966</Date>
        <ISBN>81-40-34319-4</ISBN>
        <Publisher>Dhirendra Yoga Publications</Publisher>
        <Cost currency="INR">11.50</Cost>
    </Book>
    <Book>
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN>
        <Publisher>Harper &amp;amp; Row</Publisher>
        <Cost currency="USD">2.95</Cost>
    </Book>
</BookCatalogue>
```

Если вы будете создавать проект в NetBeans (или в Idea), то XML-файл надо положить в папку с проектом — в самый корень.
Теперь сама программа для разбора и печати:

```
package edu.javacourse.xml;
```

```java

import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class DomExample {

    public static void main(String[] args) {
        try {
            // Создается построитель документа
            DocumentBuilder documentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создается дерево DOM документа из файла
            Document document = documentBuilder.parse("BookCatalog.xml");

            // Получаем корневой элемент
            Node root = document.getDocumentElement();

            System.out.println("List of books:");
            System.out.println();
            // Просматриваем все подэлементы корневого - т.е. книги
            NodeList books = root.getChildNodes();
            for (int i = 0; i < books.getLength(); i++) {
                Node book = books.item(i);
                // Если нода не текст, то это книга - заходим внутрь
                if (book.getNodeType() != Node.TEXT_NODE) {
                    NodeList bookProps = book.getChildNodes();
                    for(int j = 0; j < bookProps.getLength(); j++) {
                        Node bookProp = bookProps.item(j);
                        // Если нода не текст, то это один из параметров книги - печатаем
                        if (bookProp.getNodeType() != Node.TEXT_NODE) {
                            System.out.println(bookProp.getNodeName() + ":" + bookProp.getChildNodes().item(0).getTextC
                        }
                    }
                    System.out.println("===========>>>>");
                }
            }

        } catch (ParserConfigurationException ex) {
            ex.printStackTrace(System.out);
        } catch (SAXException ex) {
            ex.printStackTrace(System.out);
        } catch (IOException ex) {
```

```
49              ex.printStackTrace(System.out);
50          }
51      }
52  }
```

Ну что же — давайте разбираться, что и как тут работает. Для того, чтобы загрузить структуру мы должны создать объект класса **DocumentBuilder** (строка 17).
Потом загружаем текстовый XML-файл и разбираем его, получая объект **Document** (строка 19). Это и есть объектное представление всей информации внутри XML — наше дерево (сразу вспоминаются «Джентльмены удачи» — «а вон оно, дерево»).
Дальше мы начинаем «обход» нашего дерева, используя методы, которые предоставляют стандартные объекты DOM — Node, NodeList. Для того, чтобы использовать эти объекты, их честно надо изучать — смотреть, какие у них есть методы и т.д. Я предлагаю посмотреть код программы и комментарии — некоторые особенности работы с DOM вам станут понятны из кода. Если коротко — класс **Node** предназначен для любого элемента XML — текст или тэг или атрибут. Т.е. все в XML есть **Node**. Дальше идет специализация за счет наследования.
Самое главное преимущество DOM (на мой взгляд) заключается в возможности редактировать данные — ведь по сути это деревообразная структура, состоящая из унифицированных объектов, она загружена полностью в память и мы можем туда добавлять новые элементы, исправлять или удалять существующие. Также можно осуществлять поиск и делать выборку. Это дает хорошие возможности для решения конкретных задач при работе с данными. Предлагаю пример, в котором мы считаем наш XML-файл с книгами и добавим еще одну книгу в нашу структуру. После этого мы сохраним XML в файл. Самым важным тут будет метод **addNewBook** — именно в нем мы создаем унифицированные объекты, которые потом вставляются в наше дерево. Также мы используем наследника класса **Node** — класс **Element**. Этот класс предназначен именно для тэгов. Ему можно устанавливать имя и атрибуты.

```
1  package edu.javacourse.xml;
2
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import javax.xml.parsers.DocumentBuilder;
6  import javax.xml.parsers.DocumentBuilderFactory;
7  import javax.xml.parsers.ParserConfigurationException;
8  import javax.xml.transform.Transformer;
9  import javax.xml.transform.TransformerException;
10 import javax.xml.transform.TransformerFactory;
11 import javax.xml.transform.TransformerFactoryConfigurationError;
12 import javax.xml.transform.dom.DOMSource;
13 import javax.xml.transform.stream.StreamResult;
14 import org.w3c.dom.DOMException;
15 import org.w3c.dom.Document;
16 import org.w3c.dom.Element;
17 import org.w3c.dom.Node;
18 import org.xml.sax.SAXException;
19
20 public class DomExample2 {
```

```
21
22      public static void main(String[] args) {
23          try {
24              // Создается построитель документа
25              DocumentBuilder documentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
26              // Создается дерево DOM документа из файла
27              Document document = documentBuilder.parse("BookCatalog.xml");
28
29              // Вызываем метод для добавления новой книги
30              addNewBook(document);
31
32          } catch (ParserConfigurationException ex) {
33              ex.printStackTrace(System.out);
34          } catch (SAXException ex) {
35              ex.printStackTrace(System.out);
36          } catch (IOException ex) {
37              ex.printStackTrace(System.out);
38          }
39      }
40
41      // Функция добавления новой книги и записи результата в файл
42      private static void addNewBook(Document document) throws TransformerFactoryConfigurationError, DOMException {
43          // Получаем корневой элемент
44          Node root = document.getDocumentElement();
45
46          // Создаем новую книгу по элементам
47          // Сама книга <Book>
48          Element book = document.createElement("Book");
49          // <Title>
50          Element title = document.createElement("Title");
51          // Устанавливаем значение текста внутри тега
52          title.setTextContent("Incredible book about Java");
53          // <Author>
54          Element author = document.createElement("Author");
55          author.setTextContent("Saburov Anton");
56          // <Date>
57          Element date = document.createElement("Date");
58          date.setTextContent("2015");
59          // <ISBN>
60          Element isbn = document.createElement("ISBN");
61          isbn.setTextContent("0-06-999999-9");
62          // <Publisher>
63          Element publisher = document.createElement("Publisher");
64          publisher.setTextContent("Java-Course publisher");
65          // <Cost>
66          Element cost = document.createElement("Cost");
67          cost.setTextContent("499");
```

```
        // Устанавливаем атрибут
        cost.setAttribute("currency", "RUB");

        // Добавляем внутренние элементы книги в элемент <Book>
        book.appendChild(title);
        book.appendChild(author);
        book.appendChild(date);
        book.appendChild(isbn);
        book.appendChild(publisher);
        book.appendChild(cost);
        // Добавляем книгу в корневой элемент
        root.appendChild(book);

        // Записываем XML в файл
        writeDocument(document);
    }

    // Функция для сохранения DOM в файл
    private static void writeDocument(Document document) throws TransformerFactoryConfigurationError {
        try {
            Transformer tr = TransformerFactory.newInstance().newTransformer();
            DOMSource source = new DOMSource(document);
            FileOutputStream fos = new FileOutputStream("other.xml");
            StreamResult result = new StreamResult(fos);
            tr.transform(source, result);
        } catch (TransformerException | IOException e) {
            e.printStackTrace(System.out);
        }
    }
}
```

Я прекрасно понимаю, что представил не очень много информации для полного понимания и большой практики работы с XML в виде DOM, но думаю, что начало положено и вы сможете сами пойти дальше. Еще раз повторюсь — на самом деле вы просто построили дерево объектов, каждый из которых является отдельным элементом XML-документа и у вас есть инструмент, чтобы его редактировать.
Попробуйте добавить иную структуру в наше дерево — например список контактов с полями Имя, Фамилия, телефон и email. Вся необходимая информация у вас уже есть.

Сама модель DOM — это конечный результат разбора XML-строки (из файла или из какого-то другого источника — например из сети по URL). Но ведь кто-то разобрал эту строку — и самое время узнать про SAX — Simple API for XML.

SAX

SAX — это набор классов и интерфейсов, задача которых дать механизм разбора XML в строковом представлении. Основными классами я бы назвал два:

1. **SAXParser**
2. **DefaultHandler**

Я бы так описал их совместную работу: класс **SAXParser** начинает читать содержимое XML и когда наступает момент «Х» — начинается новый тэг, заканчивается тэг, начинается текст внутри тэга — он может вызывать определенную функцию класса **DefaultHandler**. Для того, чтобы «наполнить» эти вызовы со стороны **SAXParser** хоть каким-то смыслом, надо расширить класс **DefaultHandler**.
Давайте посмотрим пример печати содержимого тэга NAME из XML такого вида:

```
<?xml version="1.0" encoding="UTF-8"?>
<PHONEBOOK>
    <PERSON>
        <NAME>Joe Wang</NAME>
        <EMAIL>joe@yourserver.com</EMAIL>
        <TELEPHONE>202-999-9999</TELEPHONE>
        <WEB>www.java2s.com</WEB>
    </PERSON>
    <PERSON>
        <NAME>Karol</NAME>
        <EMAIL>karol@yourserver.com</EMAIL>
        <TELEPHONE>306-999-9999</TELEPHONE>
        <WEB>www.java2s.com</WEB>
    </PERSON>
    <PERSON>
        <NAME>Green</NAME>
        <EMAIL>green@yourserver.com</EMAIL>
        <TELEPHONE>202-414-9999</TELEPHONE>
        <WEB>www.java2s.com</WEB>
    </PERSON>
</PHONEBOOK>
```

Сама программа выглядит так:

```
package edu.gemini.xml;

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

import org.xml.sax.Attributes;
```

```java
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SaxExample {

    public static void main(String args[]) {

        // Имя файла
        final String fileName = "Phonebook.xml";

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

            // Здесь мы определили анонимный класс, расширяющий класс DefaultHandler
            DefaultHandler handler = new DefaultHandler() {
                // Поле для указания, что тэг NAME начался
                boolean name = false;

                // Метод вызывается когда SAXParser "натыкается" на начало тэга
                @Override
                public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXE
                    // Если тэг имеет имя NAME, то мы этот момент отмечаем - начался тэг NAME
                    if (qName.equalsIgnoreCase("NAME")) {
                        name = true;
                    }
                }

                // Метод вызывается когда SAXParser считывает текст между тэгами
                @Override
                public void characters(char ch[], int start, int length) throws SAXException {
                    // Если перед этим мы отметили, что имя тэга NAME - значит нам надо текст использовать.
                    if (name) {
                        System.out.println("Name: " + new String(ch, start, length));
                        name = false;
                    }
                }
            };

            // Стартуем разбор методом parse, которому передаем наследника от DefaultHandler, который будет вызываться
            saxParser.parse(fileName, handler);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Как видите, все достаточно несложно — но это на первый взгляд. Если у вас достаточно запутанный XML, то его разбор может стать весьма сложной задачей — попробуйте вложить внутрь тэга NAME еще какой-нибудь тэг с вложенным текстом перед именем **Joe Wang**. Что-то вроде такого (я привожу только кусочек, чтобы сразу было видно, что мы сделали:

```
<?xml version="1.0" encoding="UTF-8"?>
<PHONEBOOK>
    <PERSON>
        <NAME><MAIN_NAME>Noname person</MAIN_NAME>Joe Wang</NAME>
    </PERSON>
</PHONEBOOK>
```

Как вы можете догадаться (лучше конечно же убедиться, запустив программу) вместо **Joe Wang** будет напечатано **Noname person**. Что делать и как это надо исправлять — ваша домашняя задача. Как я уже упоминал — алгоритмически это может быть не так просто.

Но тем не менее почему SAX используется ? Самое главное это то, что он позволяет читать просто гигантские файлы — если вам надо при обработке только что-то найти и изредка что-то запомнить. Ведь созданием каких-либо объектов для получения результатов парсинга управляете вы сами. С одной стороны — все в ваших руках. С другой стороны — ответственность тоже на вас. Что «насобираете», то и ваше.

В моей практике наиболее часто встречалась задача импорта больших объемов данных. В одной системе подготавливается огромный XML (сотни мегабайт) и он передается в другую систему, где данные из него можно загрузить в базу. Например список фирм или транзакций по каким-то операциям. Информация об одной фирме занимает мало места и ее легко загрузить в память и потом записать в базу, но количество таких записей может выражаться сотнями миллионов. В такой ситуации SAX решает задачу очень эффективно и удобно.

StAX

В версии Java SE 6 появился еще один вариант разбора XML — Streaming API for XML (StAX). У этой технологии я хотел бы выделить два важных момента:

1. В отличии от SAX, который вызывает ваш обработчик тогда, когда считает нужным, в StAX вы сами управляете перемещением по тэгам
2. StAX позволяет не только читать, но и писать

Давайте не будем сильно заниматься копанием в особенностях StAX, а посмотрим приер, который иллюстрирует базовые основы работы StAX при чтнеии файла.

```
package edu.javacourse.xml;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

public class StaxExample {

    public static void main(String[] args) {
        final String fileName = "BookCatalog.xml";

        try {
           XMLStreamReader xmlr = XMLInputFactory.newInstance().createXMLStreamReader(fileName, new FileInputStream(fil

            while (xmlr.hasNext()) {
                xmlr.next();
                if (xmlr.isStartElement()) {
                    System.out.println(xmlr.getLocalName());
                } else if (xmlr.isEndElement()) {
                    System.out.println("/" + xmlr.getLocalName());
                } else if (xmlr.hasText() && xmlr.getText().trim().length() > 0) {
                    System.out.println("    " + xmlr.getText());
                }
            }
        } catch (FileNotFoundException | XMLStreamException ex) {
            ex.printStackTrace();
        }
    }
}
```

Как видите, все достаточно просто — вы создаете объект класса **XMLStreamReader**, который позволяет вам перемещаться последовательно по всем элементам XML и проверять, что за элемент вы нашли. У класса **XMLStreamReader** есть ряд методов, которые позволяют получить доступ к свойствам элемента XML — имя, текст, атрибуты. Что с этой информацией делать — опять же зависит от вашей задачи.
Самый главный вопрос — а зачем все это надо ? Ответ такой же, как и для SAX — обработка больших объемов данных.

Теперь пример, как можно использовать StAX для записи файла. Что здесь хорошего — можно писать большие файлы. Что же касается удобства — на мой взгляд не самый удобный способ, хотя в нем есть своя элегантность и целостность. Пример достаточно несложный — он пишет в XML несколько книг. Смотрим.

```
package edu.javacourse.xml;

import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
```

```java
import java.util.Date;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

/**
 * Пример записи XML-файла с помощью технологии StAX
 *
 * @author ASaburov
 */
public class StaxWriteExample {

    public static void main(String[] args) {
        try {
            XMLOutputFactory output = XMLOutputFactory.newInstance();
            XMLStreamWriter writer = output.createXMLStreamWriter(new FileWriter("result.xml"));

            // Открываем XML-документ и Пишем корневой элемент BookCatalogue
            writer.writeStartDocument("1.0");
            writer.writeStartElement("BookCatalogue");
            // Делаем цикл для книг
            for (int i = 0; i < 5; i++) {
                // Записываем Book
                writer.writeStartElement("Book");

                // Заполняем все тэги для книги
                // Title
                writer.writeStartElement("Title");
                writer.writeCharacters("Book #" + i);
                writer.writeEndElement();
                // Author
                writer.writeStartElement("Author");
                writer.writeCharacters("Author #" + i);
                writer.writeEndElement();
                // Date
                writer.writeStartElement("Date");
                writer.writeCharacters(new SimpleDateFormat("yyyy-MM-dd").format(new Date()));
                writer.writeEndElement();
                // ISBN
                writer.writeStartElement("ISBN");
                writer.writeCharacters("ISBN #" + i);
                writer.writeEndElement();
                // Publisher
                writer.writeStartElement("Publisher");
                writer.writeCharacters("Publisher #" + i);
                writer.writeEndElement();
                // Cost
```

```
                writer.writeStartElement("Cost");
                writer.writeAttribute("currency", "USD");
                writer.writeCharacters("" + (i+10));
                writer.writeEndElement();

                // Закрываем тэг Book
                writer.writeEndElement();
            }
            // Закрываем корневой элемент
            writer.writeEndElement();
            // Закрываем XML-документ
            writer.writeEndDocument();
            writer.flush();
        } catch (XMLStreamException | IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

В общем мы рассмотрели основные варианты разбора XML, которые предоставляет Java. Отмечу еще раз — цель всех этих технологий — разобрать строку на отдельные элементы. Больше ничего. Есть еще одна интересная (и востребованная) технология — JAXB (Java Architecture for XML Binding). Я отложу ее рассмотрение до другого раза. А пока займемся следующим функционалом — поиском по сформированному дереву.

Прежде чем мы начнем рассмотрение этого вопроса, предлагаю вам попробовать написать код для поиска названия книги у которой должен быть определенный автор и цена в рублях (заранее занесите такого в XML). Используйте DOM для построения полного дерева элементов и уже для него вам надо написать код. Это не самое простое занятие, но зато попрактикуетесь.

Поиск по XML — XPath

Для осуществления поиска нужных вам данных был разработан специальный язык — XPath. Я не ставлю себе цель научить вас всем тонкостям этого языка — я предлагаю посмотреть как можно его использовать для того, чтобы найти нужную вам информацию. Самым правильным решением на мой взгляд (как всегда) является пример использования. Вот давайте его и посмотрим. Пример демонстрирует, как можно найти ту или иную книгу (книги) по определенным критениям.

```
package edu.javacourse.xml;

import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
```

```java
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class XPathExample {

    public static void main(String[] args) {
        try {
            DocumentBuilder documentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
            Document document = documentBuilder.parse("BookCatalog.xml");

            printCost(document);
            printCost2(document);
            printCost3(document);
            printCost4(document);
            printCost5(document);

        } catch (XPathExpressionException | ParserConfigurationException | SAXException | IOException ex) {
            ex.printStackTrace(System.out);
        }
    }

    // Печать всех элементов Cost
    private static void printCost(Document document) throws DOMException, XPathExpressionException {
        System.out.println("Example 1 - Печать всех элементов Cost");
        XPathFactory pathFactory = XPathFactory.newInstance();
        XPath xpath = pathFactory.newXPath();

        // Пример записи XPath
        // Подный путь до элемента
        //XPathExpression expr = xpath.compile("BookCatalogue/Book/Cost");
        // Все элементы с таким именем
        //XPathExpression expr = xpath.compile("//Cost");
        // Элементы, вложенные в другой элемент
        XPathExpression expr = xpath.compile("//Book/Cost");

        NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i);
            System.out.println("Value:" + n.getTextContent());
        }
```

```
55          System.out.println();
56      }
57
58      // Печать элемента Cost у которого атрибут currency='USD'
59      private static void printCost2(Document document) throws DOMException, XPathExpressionException {
60          System.out.println("Example 2 - Печать элемента Cost у которого атрибут currency='USD'");
61          XPathFactory pathFactory = XPathFactory.newInstance();
62          XPath xpath = pathFactory.newXPath();
63          XPathExpression expr = xpath.compile("BookCatalogue/Book/Cost[@currency='USD']");
64          NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
65          for (int i = 0; i < nodes.getLength(); i++) {
66              Node n = nodes.item(i);
67              System.out.println("Value:" + n.getTextContent());
68          }
69          System.out.println();
70      }
71
72      // Печать элементов Book у которых значение Cost > 4
73      private static void printCost3(Document document) throws DOMException, XPathExpressionException {
74          System.out.println("Example 3 - Печать элементов Book у которых значение Cost > 4");
75          XPathFactory pathFactory = XPathFactory.newInstance();
76          XPath xpath = pathFactory.newXPath();
77          XPathExpression expr = xpath.compile("BookCatalogue/Book[Cost>4]");
78          NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
79          for (int i = 0; i < nodes.getLength(); i++) {
80              Node n = nodes.item(i);
81              System.out.println("Value:" + n.getTextContent());
82          }
83          System.out.println();
84      }
85
86      // Печать первого элемента Book
87      private static void printCost4(Document document) throws DOMException, XPathExpressionException {
88          System.out.println("Example 4 - Печать первого элемента Book");
89          XPathFactory pathFactory = XPathFactory.newInstance();
90          XPath xpath = pathFactory.newXPath();
91          XPathExpression expr = xpath.compile("BookCatalogue/Book[2]");
92          NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
93          for (int i = 0; i < nodes.getLength(); i++) {
94              Node n = nodes.item(i);
95              System.out.println("Value:" + n.getTextContent());
96          }
97          System.out.println();
98      }
99
100     // Печать цены книги у которой Title начинается с Yogasana
101     // Варианты доступа к относительным узлам:
```

```
    // ancestor , ancestor-or-self, descendant, descendant-or-self
    // following, following-sibling, namespace, preceding, preceding-sibling
    private static void printCost5(Document document) throws DOMException, XPathExpressionException {
        System.out.println("Example 5 - Печать цены книги у которой Title начинается с 'Yogasana'");
        XPathFactory pathFactory = XPathFactory.newInstance();
        XPath xpath = pathFactory.newXPath();
        XPathExpression expr = xpath.compile("BookCatalogue/Book/Cost"
                + "[starts-with(preceding-sibling::Title, 'Yogasana')"
                + " or "
                + "starts-with(following-sibling::Title, 'Yogasana')]");
        NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i);
            System.out.println("Value:" + n.getTextContent());
        }
        System.out.println();
    }

}
```

Как легко можно увидеть, в каждой функции для работы с XPath используется одна и та же последовательность действий.

```
    // Создать XPathFactory
    XPathFactory pathFactory = XPathFactory.newInstance();
    // Создать XPath
    XPath xpath = pathFactory.newXPath();
    // Получить скомпилированный вариант XPath-выражения
    XPathExpression expr = xpath.compile("BookCatalogue/Book[2]");
    // Применить XPath-выражение к документу для поиска нужных элементов
    NodeList nodes = (NodeList) expr.evaluate(document, XPathConstants.NODESET);
```

Еще раз проговорю главную цель XPath — надо находить информацию по определенным критериям. Рассматривайте XPath именно с этой позиции и вы получите очень удобный и эффективный инструмент для работы с XML.

Что касается изучения непосредственно XPath, то могу рекомендовать следующие сайты:

- XPath tutorial
- XPath Overview
- Начало работы с XPath

Проверка

Когда вы начинаете работать с какой-либо структурой, то важным моментом является ее правильное заполнение. Для XML это включает целый ряд задач — правильное написание самого XML, указание только определенных тэгов, взаимное расположение тэгов (например один тэг может быть только внутри другого, а не наоборот), набор атрибутов, значения полей и атрибутов. Все это весьма важно.
Если мы договариваемся передавать определенную информацию между двумя системами, то они должны заранее знать, какие именно тэги и с какой информацией будут включаться в XML. Постарайтесь осознать важность проверки корректности заполнения информации — ведь в реальную систему нередко приходят миллионы сообщений и отбрасывать некорректные — благородная и ответственная задача. Для этой цели было разработано механизмов, о которых мы вкратце и поговорим.

В первую очередь XML должен быть well-formed — «правильно сформированный». Это значит, что все тэги должны «закрываться» (подобно скобкам в математическом выражении), закрытие и открытие тэгов не должно пересекаться и т.д. Более подробно можно посмотреть анпример здесь: Well-formed document.
Но кроме просто правильно сформированного XML есть потребность более четкого (жесткого) определения структуры XML-документа. И эта потребность имеет свое решение.
В самом начале работы над XML был предложен механизм DTD — Document Type Definition. Это описание структуры XML-документа — какие тэги допустимы, в каком порядке, какой тэг является вложенным и для какого, набор атрибутов и прочая. Для знакомства можно почитать эту статью: Document type definition. На сегодня этот вариант считается устаревшим и на его смену пришла XML-схема — XML Schema.
XML Schema представляет собой файл (чаще всего с расширением XSD — XML Shema Definition), который с форме XML (да-да, именно так — описываем правила формирования XML с помощью XML) описывает что и в каком виде может быть включено в XML-документ. XML Schema является весьма любопытным объектом для изучения и я настоятельно рекомендую почитать об этой технологии. В ней есть очень много моментов, про которые надо почитать отдельно от какого-либо языка. Вот несколько источников:

- XML Schema Tutorial.
- «The Art of XSD. SQL Server XML Schema Collections» Jacob Sebastian — мне эта книга очень понравилась

Но цель этой статьи состоит в том, чтобы показать, как можно использовать XSD в Java. Так что предлагаю посмотреть пример, как надо сделать проверку XML по указанному XSD. Сначала приведу XSD и XML для примера.
XSD-файл, который описывает правила формирования XML — здесь мы описали формат сообщения, которое предположим может быть послано от гражданина в какой-нибудь департамент (основано на вполне реальном файле).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="http://www.java-course.ru/xml/message"
3              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4              xmlns:tns="http://www.java-course.ru/xml/message">
5
6      <xsd:complexType name="Message">
7          <xsd:all>
8              <xsd:element minOccurs="1" maxOccurs="1" name="messagedate" type="xsd:dateTime"/>
9              <xsd:element minOccurs="1" maxOccurs="1" name="surname" type="xsd:string"/>
10             <xsd:element minOccurs="1" maxOccurs="1" name="firstname" type="xsd:string"/>
11             <xsd:element minOccurs="1" maxOccurs="1" name="patronymic" type="xsd:string"/>
```

```
            <xsd:element minOccurs="1" maxOccurs="1" name="postaladdress" type="xsd:string"/>
            <xsd:element minOccurs="1" maxOccurs="1" name="email" type="xsd:string"/>
            <xsd:element minOccurs="1" maxOccurs="1" name="department" type="xsd:string"/>
            <xsd:element minOccurs="1" maxOccurs="1" name="text" type="xsd:string"/>
            <xsd:element minOccurs="0" maxOccurs="1" name="filename" type="xsd:string"/>
            <xsd:element name="filedata" type="xsd:base64Binary" minOccurs="0" maxOccurs="1"/>
        </xsd:all>
    </xsd:complexType>

    <xsd:element name="message" type="tns:Message"/>

</xsd:schema>
```

А это наш XML с корректной информацией (с заполненным сообщением)

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

<jc:message xmlns:jc="http://www.java-course.ru/xml/message"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.java-course.ru/xml/message Message.xsd">

    <messagedate>2013-05-04T18:13:51+04:00</messagedate>
    <surname>Петров</surname>
    <firstname>Семен</firstname>
    <patronymic>Васильевич</patronymic>
    <postaladdress>Санкт-Петербург, ул.Зенитчиков, д.23, кв. 59</postaladdress>
    <email>petrov@pisem.net</email>
    <department>Департамент здравоохранения</department>
    <text>Сообщение, по поводу которого мы должны согласовать наши позиции</text>
    <filename>svid.txt</filename>
    <filedata>0J/RgNC40LzQtdGA</filedata>

</jc:message>
```

Ну и наконец Java-код, который позволяет проверить корректность XML по его XSD.

```
package edu.javacourse.xml;

import java.io.File;
import java.io.IOException;
import javax.xml.XMLConstants;
```

```java
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

import org.xml.sax.SAXException;

public class XsdValidator
{
    public static void main(String[] args)
    {
        boolean answer = validateXMLSchema("Message.xsd", "Message.xml");
        System.out.println("Result:" + answer);
    }

    public static boolean validateXMLSchema(String xsdPath, String xmlPath)
    {
        try {
            // Получить фабрику для схемы
            SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
            // Загрузить схему из XSD
            Schema schema = factory.newSchema(new File(xsdPath));
            // Создать валидатор (проверялбщик)
            Validator validator = schema.newValidator();
            // Запусить проверку - если будет исключение, значит есть ошибки.
            // Если нет - все заполнено правильно
            validator.validate(new StreamSource(new File(xmlPath)));
        } catch (IOException | SAXException e) {
            System.out.println("Exception: " + e.getMessage());
            return false;
        }
        return true;
    }
}
```

Теперь можно «поиграть» с нашими данными и посмотреть, как реагирует наша программа на изменения в данных. Я бы рекомендовал попробовать создать свой вариант XML и XSD и использовать нашу программу для их проверки.

Преобразование

Необходимость преобразования XML во что-то другое лежала на поверхности и ее реализовали. Эту функцию решает технология XSLT (Extensible Stylesheet Language Transformations). У меня нет цели в этой статье углубляться в сам XSLT — для этого можно посмотреть в интернете неплохие статьи — например:

- XSLT Tutorial
- XSL / XSLT Intro and Resources

Я только остановлюсь на вариантах использования, которые логически вытекают из самого XML.
Как я уже говорил, XML весьма удобен для хранения и передачи информации в структурированном виде. Но для отображения это далеко не самый лучший вариант. Т.е. само преобразование требуется несомненно. Кроме того учитывая, что сами данные в XML могут предназначаться разным системам, надо иметь возможность гибко настраивать такое преобразование для каждой системы. Например, информация о тех же книгах для десктопного компьютера может быть представлена в одном виде, а для мобильного телефона — совсем в другом (более компактном). Информация в одном экземпляре, а ее представление — во многих. Удобно.
Мало того, с учетом все возрастающей мощности клиентских компьютеров/планшетов/смартфонов появляется возможность возложить ношу преобразования на них — передаем им XML и файл для преобразования (XSL-файл) и пусть они сами его преобразовывают. Это может серьезно снизить нагрузку на серверную сторону и тем самым повысить общую производительность.

А теперь посмотрим пример преобразования каталога книг в HTML-таблицу. Здесь важно увидеть, что информация для преобразования может быть в любой момент изменена без изменения кода. Комментарии смотрите в самом коде.
XML с данными о книгах

```xml
<?xml version="1.0" encoding="UTF-8"?>
<BookCatalogue>
    <Book>
        <Title>Yogasana Vijnana: the Science of Yoga</Title>
        <Author>Dhirendra Brahmachari</Author>
        <Date>1966</Date>
        <ISBN>81-40-34319-4</ISBN>
        <Publisher>Dhirendra Yoga Publications</Publisher>
        <Cost currency="INR">11.50</Cost>
    </Book>
    <Book>
        <Title>The First and Last Freedom</Title>
        <Author>J. Krishnamurti</Author>
        <Date>1954</Date>
        <ISBN>0-06-064831-7</ISBN>
        <Publisher>Harper &amp;amp; Row</Publisher>
        <Cost currency="USD">2.95</Cost>
    </Book>
</BookCatalogue>
```

XSL для преобразования в HTML-таблицу

```
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output omit-xml-declaration="yes"/>

    <xsl:template match="/">
        <html>
            <body>
                <table border="1">
                    <thead>
                        <tr>
                            <th>Title</th>
                            <th>Author</th>
                            <th>Cost</th>
                        </tr>
                    </thead>
                    <xsl:for-each select="BookCatalogue/Book">
                        <tr>
                            <xsl:call-template name="PrintBook"/>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template name="PrintBook">
        <td>
            <xsl:value-of select="Title"/>
        </td>
        <td>
            <xsl:value-of select="Author"/>
        </td>
        <td>
            <xsl:value-of select="Cost"/>
        </td>
    </xsl:template>

</xsl:stylesheet>
```

Ну и сама программа для запуска преобразования.

```
```

```
package edu.javacourse.xml;

```

```java
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class XslConverter
{
    public String xmlToString(String xmlFile, String xslFile) throws Exception {
        // Открыть файлы в виде потоков
        InputStream xml = new FileInputStream(xmlFile);
        InputStream xsl = new FileInputStream(xslFile);
        // Сщоздать источник для транформации из потоков
        StreamSource xmlSource = new StreamSource(xml);
        StreamSource stylesource = new StreamSource(xsl);

        // Создать байтовый поток для результата
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        // COздать приемноик для результатат из байтового потока
        StreamResult xmlOutput = new StreamResult(bos);
        // Создать трансформатор и выполнить трансформацию
        Transformer transformer = TransformerFactory.newInstance().newTransformer(stylesource);
        transformer.transform(xmlSource, xmlOutput);

        // вернуть результат в виде строки
        return bos.toString();
    }

    public static void main(String[] arg) throws IOException {
        XslConverter c = new XslConverter();

        final String xml = "BookCatalog.xml";
        final String xsl = "BookCatalog.xsl";
        try {
            String result = c.xmlToString(xml, xsl);
            System.out.println(result);
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

В качестве домашнего задания — попробуйте сделать систему, которая может выбирать один из нескольких XSL по аргументу, который передается в функцию **xmlToString**.

И теперь нас ждет следующая статья: Reflection — основы.

30 comments to *Работаем с XML*

- 
  Август 19, 2015 at 12:42
  Евген says:

  Недавно разбирался с методами парсинга XML в Java. Пробовал и вышеописанные DOM и SAX — но больше понравилось работать с JAXB. В разы проще, и удобнее. В целом ресурс у Вас неплохой, подключу к изучению)

  Reply

  - 
    Август 19, 2015 at 17:26
    admin says:

    JAXB все-таки достаточно специфическая вещь — я хотел ее оставить на потом. Хотя смотрится JAXB конечно красиво и элегантно.

    Reply

- 
  Август 26, 2015 at 11:51
  Полина says:

  Огромное спасибо за хороший материал. Очень жду, когда будут коллекции

  Reply

- Январь 29, 2016 at 19:35
  Alexey says:

  Спасибо за труд, хорошая работа! Пригодиться.

  Reply
- 
  Август 20, 2016 at 12:20
  Grif says:

  … во фразе «Harper & Row» надо заменить & на & .

  Reply
  - Август 22, 2016 at 14:43
    admin says:

    Исправил

    Reply
    - Сентябрь 9, 2016 at 07:27
      Денис says:

      В последнем примере не исправлено. Приводит к ошибке.

      Reply

Сентябрь 9, 2016 at 10:22
admin says:

Спасибо — исправил там тоже.

Reply

Ноябрь 19, 2016 at 23:04
Анатолий says:

//StAx
В коде ошибка.
Должно быть примерно так:

// Открываем XML-документ и Пишем корневой элемент BookCatalogue
writer.writeStartDocument(«UTF-8», «1.0»);
writer.writeStartElement(«BookCatalogue»);

Reply

Ноябрь 21, 2016 at 11:53
admin says:

Спасибо. Исправил.

Reply

Июнь 28, 2017 at 09:59

Yury says:

Спасибо за пример Dom парсинга. Подскажите пожалуйста как будет выглядеть участок кода для вывода возможных атрибутов в дочерних тегах, например
Dhirendra Brahmachari

Reply

- 
  Июнь 28, 2017 at 12:44
  admin says:

  Вопрос очень неконкретный — что значит «вывод возможных атрибутов в дочерних тегах» ?

  Reply

- 
  Июнь 30, 2017 at 11:32
  Yury says:

  Разобрался: Если теги имеют атрибуты то примерно так
  System.out.println(«Атрибуты :» +
  bookProp.getAttributes().getNamedItem(«name») +
  bookProp.getAttributes().getNamedItem(«url») +
  bookProp.getAttributes().getNamedItem(«description»));

  Reply

- 
  Июль 11, 2017 at 17:19
  Роман says:

  Очень хороший материал для изучения и поиска решений типовых задач.
  Можно узнать когда появится подобная статья про JAXB?

А также может у автора найдётся время и для похожей статьи разбора JSON?
Заранее благодарен.

Reply

- 

  Июль 12, 2017 at 10:51
  admin says:

  Я пока взял паузу — много идей, много работы и очень мало свободного времени.

  Reply

Июль 18, 2017 at 17:44
Михаил says:

А как с помощью StAX создать хмл с красивой структурой, как у в ваших примерах, нужно прописывать новую строку и пробелы вручную в коде? Ибо если использовать приведенный вами код для записи нового файла, то он будет полностью записан в виде одной строки, что визуально вообще не воспринимается.

Reply

- 

  Июль 19, 2017 at 10:26
  admin says:

  Я не пробовал так делать — в принципе это не проблема — загружаете в редактор и делаете форматирование — обычно все становится красиво.
  Тем более, что в общем-то XML не для просмотра человеком, а для обработки программой. И ей лишние пробелы только мешают.
  Если подходить более тщательно, то в XML важны и пробелы тоже — поэтому красивый XML не всегда является правильным по своей природе.

  Reply

Июль 20, 2017 at 17:47
Михаил says:

Да, я оставил всё как есть, спасибо за ответ. (просто немного перфекционист в душе)

Reply

Май 31, 2018 at 10:43
Дмитрий says:

Добрый день!
Я пишу приложение, в котором применил разбор XML из этой статьи по методу DOM.
При создании класса для разбора, компилятор ругался, что не может найти файл CatalogList.xml. Мой xml файл лежит не в корне проекта, а в пакете. Я написал ему полный путь до файла начиная с src: «src/catalog/DAO/CatalogList.xml». NetBeans это устроило и он начал парсить.
Сейчас заметил, что при запуске файла jar отдельно от NetBeans, ничего не происходит, приложение не запускается.
Попробовал через cmd. Выдаёт FileNotFoundException для этого xml файла.
Я открыл jar как архив и увидел, что в нём и правда нет папки «src». Но если не указать «src/» в пути к файлу, то тогда NetBeans не может его найти.
Попробовал написать путь как просто «CatalogList.xml», это удовлетворило NetBeans, но cmd по прежнему выдаёт то же исключение.

Reply

Май 31, 2018 at 11:16
Дмитрий says:

Получилось!
Почитал, что если файл пакуется в jar, то нужно его вызывать в коде через getClass().getResource(url).
Было:

```
DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse("CatalogList.xml");
```

Стало:

```
ClassLoader cl = this.getClass().getClassLoader();
InputStream f = cl.getResourceAsStream("CatalogList.xml");
DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(f);
```

Запустил приложение и увидел, что отвалились все текстовые файлы. Видимо их я тоже вызывал не через ресурсы.

Reply

- 
  Май 31, 2018 at 11:18
  admin says:

  Путь до файла — это достаточно любпытная штука. Во-первых — укажите просто имя файла. Без каких-либо путей. Тогда в случае запуска из под NetBeans его можно положить в каталог проекта — в самый корень.

  При запуске из командной строки можно поместить файл в тот же каталог, откдуа запускаете проект. Но тут есть тонкости — хотелось бы посмотреть на строку запуска и откуда запускаете.

  Reply

  - 
    Май 31, 2018 at 11:47
    Дмитрий says:

    Не думаю, что верно понял вопрос. Строка запуска из командной строки:

    ```
    java -jar "CatalogMail.jar"
    ```

Перед этим вызываю команду cd до папки проекта\dist, в которой лежит jar файл.

Кстати говоря, с txt файлами пока не разобрался. Там файл вызывается через ресурс, то есть проблема должна быть в другом. А командная строка выдаёт FileNotFoundException (Синтаксическая ошибка в имени файла, имени папки или метке тома).

Reply

Май 31, 2018 at 12:10
Дмитрий says:

Изменил

```
1 File textFile = new File(cl.getResource(txtPath).getFile());
```

на

```
1 InputStream textFile = cl.getResourceAsStream(txtPath);
```

И исключение перестало кидаться, но текст превратился в набор кракозябр. Через NetBeans всё нормально по-русски.
В чём может быть дело? Нужно ли как-то закрывать InputStream?

Reply

Май 31, 2018 at 13:09
admin says:

Надо открывать текстовый файл с указанием кодировки. В этом случае есть смысл посмотреть — https://www.youtube.com/watch?v=RxacJWxER0w

Май 31, 2018 at 15:34
Дмитрий says:

Посмотрел видео, попробовал сделать как в нём. FileNotFoundException для txt файлов.
Немного изменил, теперь так:

```
1  private String getTextFile(String txtPath){
2          StringBuilder result = new StringBuilder();
3          try (InputStreamReader fr =
4                  new InputStreamReader(new FileInputStream(new File(cl.getResource(txtPath).toUR
5                  BufferedReader br = new BufferedReader(fr)) {
6              String line;
7              while ((line = br.readLine()) != null) {
8                  result.append(line).append("\n");
9              }
10
11         } catch (IOException ex) {
12             ex.printStackTrace(System.out);
13         } catch (URISyntaxException ex) {
14             ex.printStackTrace(System.out);
15         }
16         return result.toString();
17     }
```

Снова через NetBeans всё хорошо, а jar файл отдельно выдаёт исключение java.lang.IllegalArgumentException: URI is not hierarchical.

Май 31, 2018 at 16:49
Дмитрий says:

Всё, разобрался! Как всегда всё получилось в последние минуты рабочего дня.
Вот, что сработало для меня. И буквы русские и файлы не отвалились.

```
try (InputStreamReader fr =
                new InputStreamReader(cl.getResourceAsStream(txtPath),"UTF-8");
                BufferedReader br = new BufferedReader(fr)) {
            ...
            }
```

Август 12, 2018 at 13:35
Владимир says:

Здравствуйте. Есть решения более простые, без коллекций, например?

DefaultHandler handler = new DefaultHandler() {
Stack stack= new Stack();

// Метод вызывается когда SAXParser «натыкается» на начало тэга

@Override
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
stack.push(qName);
}

// Метод вызывается когда SAXParser считывает текст между тэгами
@Override
public void characters(char ch[], int start, int length) throws SAXException {
if(stack.peek().equalsIgnoreCase(«NAME»)) {
System.out.println(«Name: » + new String(ch, start, length));
}
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
stack.pop();
}
};

П.С. Подобную задачу решал ранее, потратил около недели) Не смог реализовать без коллекции.

Reply

- 
  Август 14, 2018 at 04:31
  admin says:

  Спасибо за другое решение. Правда стек — это тоже коллекция, просто у нее иные правила игры. Но все равно достаточно элегантное решение со стеком — это правда.

  Reply

- 
  Август 17, 2018 at 13:20
  Владимир says:

  Здравствуйте. Почему строка: XPathExpression expr = xpath.compile(«BookCatalogue/Book[1]»); работает, как надо, выводя одну книгу, а если заменить её на строку:
  XPathExpression expr = xpath.compile(«BookCatalogue/Book/Cost[1]»);
  То на выходе получается два элемента «Cost»?

  Reply

  - 
    Август 19, 2018 at 07:11
    admin says:

    Так запрос идет — первые элементы ЦЕНА у ВСЕХ книг.

    Reply

Leave a reply

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Имя *

E-mail *

Сайт

шесть + восемь =

Add comment