**XML парсеры**

**Java DOM Parser**

DOM (Document Object Model)

DOM-обработчик считывает сразу весь XML и сохраняет его, создавая иерархию в виде дерева. По дереву можно двигаться и получать доступ к нужным элементам.

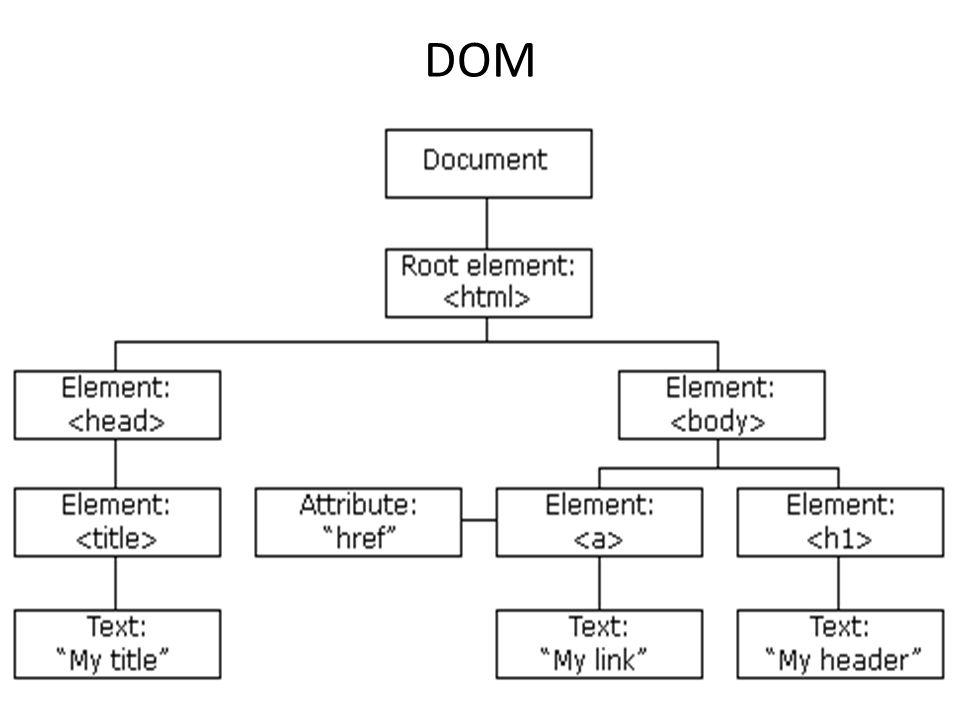


Рисунок 1. Пример дерева элементов

Имея ссылку на верхний элемент, можно получить ссылки на все его внутренние элементы. Внешний элемент называется *родительским*, его внутренние элементы называются *дочерними* элементами.

Все классы в DOM реализуют один базовый интерфейс – Node (узел, базовый для всех остальных интерфейсов). Поэтому самый частый тип данных в DOM – это Node, который может быть всем.

Методы Node для извлечения информации:

1. getNodeName – получить имя узла.
2. getNodeValue – получить значение узла.
3. getNodeType – получить тип узла.
4. getParentNode – получить узел, внутри которого находится данный узел.
5. getChildNodes – получить все производные узлы (узлы, которые внутри данного узла).
6. getAttributes – получить все атрибуты узла.
7. getOwnerDocument – получить документ этого узла.
8. getFirstChild/getLastChild – получить первый/последний производный узел.
9. getLocalName – при обработке пространств имён получает имя без префикса. Если в DocumentFactory не использовался метод setNamespaceAware(true), чтобы запустить обработку пространств имён, то это метод всегда будет возвращать null.
10. getTextContent – возвращает весь текст внутри элемента и всех элементов внутри данного элемента, включая переносы строчек и пробелы.

Элементом Node может быть как элемент, так и атрибут. НО:

1. Атрибуты есть ТОЛЬКО у элементов.
2. У элементов НЕТ значения.
3. Имя узла-элемента совпадает с именем тега, а узла-атрибута с именем атрибута.

**Задача 1**. Извлечь информацию обо всех сотрудниках из файла xml\_file1.xml и вывести её в консоль.

Пусть XML файл имеет следующее содержание.

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

<company>

<name>IT-Heaven</name>

<offices>

<office floor="1" room="1">

<employees>

<employee name="Maksim" job="Middle Software Developer" />

<employee name="Ivan" job="Junior Software Developer" />

<employee name="Franklin" job="Junior Software Developer" />

</employees>

</office>

<office floor="1" room="2">

<employees>

<employee name="Herald" job="Middle Software Developer" />

<employee name="Adam" job="Middle Software Developer" />

<employee name="Leroy" job="Junior Software Developer" />

</employees>

</office>

</offices>

</company>

Для хранения содержимого элемента ***employee*** создайте POJO. Класс ***Employee*** должен содержать два поля ***name***, ***job***, гетеры и конструктор с параметрами для задания значения этим полям.

Для хранения всех сотрудников можно использовать коллекцию **private** **static** ArrayList<Employee> employees = **new** ArrayList<>();

Перейдем к описанию кода. Для создания дерева элементов XML-документа необходимо сначала создать экземпляр класса javax.xml.parsers.DocumentBuilderFactory. Фабрика позволяет создать билдер – элемент класса javax.xml.parsers.DocumentBuilder. Элемент билдер создает **Document** – само DOM-дерево.

// Получение фабрики. По фабрике получаем билдер документов.

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

// Получение билдера из фабрики. Билдер будет парсить XML и создаст структуру Document в виде иерархического дерева.

DocumentBuilder builder = factory.newDocumentBuilder();

// При парсинге XML создается структура Document. Данный document предоставляет доступ ко всем элементам.

Document document = builder.parse(**new** File("resource/xml\_file1.xml"));

Дале опишем извлечение элементов employee и информации о них.

// getDocumentElement возвращает ROOT-элемент XML файла.

// метод getElementsByTagName("employee") возвращает список всех элементов employee внутри корневого элемента

NodeList employeeElements = document.getDocumentElement().getElementsByTagName("employee");

// Просмотр всех элементов employee из полученного списка

**for** (**int** i = 0; i < employeeElements.getLength(); i++) {

Node employee = employeeElements.item(i);

// Получение атрибутов элемента

NamedNodeMap attributes = employee.getAttributes();

// Добавление сотрудника.

// Так как атрибут это тоже Node, то можно получить значение атрибута с помощью метода getNodeValue()

employees.add(**new** Employee(attributes.getNamedItem("name").getNodeValue(), attributes.getNamedItem("job").getNodeValue()));

}

**Задача 2**. Считать с консоли имя элемента, про который нужно вывести информацию и имена всех его дочерних элементов и их атрибутов. Использовать следующий XML файл – xml\_file2.xml:

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

<root>

<oracle>

<connection value="jdbc:oracle:thin:@10.220.140.48:1521:test1" />

<user value="secretOracleUsername" />

<password value="111" />

</oracle>

<mysql>

<connection value="jdbc:mysql:thin:@10.220.140.48:1521:test1" />

<user value="secretMySQLUsername" />

<password value="222" />

</mysql>

</root>

Алгоритм работы.

1. Считать имя. Будем хранить имя в строке newElement
2. Получить элемент по имени.
3. Пройти по всем дочерним узлам: пройти по всем дочерним узлам всех дочерних узлов, которые являются элементами.

Начнем пояснение со второго пункта алгоритма.

// Получение структуры документа.

Document document = builder.parse(**new** File("resource/xml\_file2.xml"));

// Результат - список элементов. В данном случае, для удобства, рассматривается только первое совпадение в документе.

// Поиск ведется внутри всего документа, а не рут элемента. Поэтому рут элемент тоже может быть найден.

NodeList matchedElementsList = document.getElementsByTagName(newElement);

// Если элемента нет, то будет возвращаться пустой список.

// Проверка на наличие элементов в списке.

**if** (matchedElementsList.getLength() == 0) {

System.out.println("Тег " + newElement + " не был найден в файле.");

} **else** {

// Получение первого элемента.

Node foundedElement = matchedElementsList.item(0);

System.out.println("Элемент был найден!");

// Если элемент имеет данные внутри, то вызываем для него рекурсивный метод получения всей информации

**if** (foundedElement.hasChildNodes())

printInfoAboutAllChildNodes(foundedElement.getChildNodes());

}

Поясним третий пункт алгоритма.

/\*\*

\* Рекурсивный метод, который выводит информацию про все узлы внутри всех узлов.

\* @param list Список узлов.

\*/

**private** **static** **void** printInfoAboutAllChildNodes(NodeList list) {

**for** (**int** i = 0; i < list.getLength(); i++) {

Node node = list.item(i);

// У элементов есть два вида узлов - другие элементы или текстовая информация. Потому нужно разбирать две ситуации отдельно.

**if** (node.getNodeType() == Node.TEXT\_NODE) {

// Фильтрация информации, так как начальные и конечные пробелы и переносы строк не нужны. Это не информация.

String textInformation = node.getNodeValue().replace("\n", "").trim();

**if**(!textInformation.isEmpty())

System.out.println("Внутри элемента найден текст: " + node.getNodeValue());

}

// Если это не текст, а элемент, то его нужно обрабатывать как элемент.

**else** {

System.out.println("Найден элемент: " + node.getNodeName() + ", его атрибуты:");

// Получение атрибутов

NamedNodeMap attributes = node.getAttributes();

// Вывод информации о полученных атрибутах.

**for** (**int** k = 0; k < attributes.getLength(); k++)

System.out.println("Имя атрибута: " + attributes.item(k).getNodeName() + ", его значение: " + attributes.item(k).getNodeValue());

}

// Если у данного элемента еще остались узлы, то вывести всю информацию обо всех его узлах.

**if** (node.hasChildNodes())

printInfoAboutAllChildNodes(node.getChildNodes());

}

}

Результатом является список элементов. В примере рассматривается только первое совпадение.

**Запись в XML-документ**

**Задача 3**. Создать XML файл средствами Java.

Сначала необходимо создать пустой XML документ с помощью DOM-парсера. XML документ создается с помощью объекта javax.xml.parsers.DocumentBuilder, полученного из фабрики javax.xml.parsers.DocumentBuilderFactory.

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

factory.setNamespaceAware(true);

Document doc = factory.newDocumentBuilder().newDocument();

Дополнительно у фабрики можно включить поддержку пространства имен с помощью метода setNamespaceAware для работы с XML-схемой (XSD).

Основная работа с деревом DOM приходится на элемент интерфейса org.w3c.dom.Element. Ему можно добавлять дочерние элементы через метод appendChild или устанавливать атрибуты посредством метода setAttribute.

Добавление в XML документ корневого элемента ***root***.

Element root = doc.createElement("root");

root.setAttribute("xmlns", "http://www.javacore.ru/schemas/");

doc.appendChild(root);

Далее в корневой элемент добавляются дочерние элементы ***item***.

Element item1 = doc.createElement("item");

item1.setAttribute("val", "1");

root.appendChild(item1);

Element item2 = doc.createElement("item");

item2.setAttribute("val", "2");

root.appendChild(item2);

Element item3 = doc.createElement("item");

item3.setAttribute("val", "3");

root.appendChild(item3);

**Java SAX Parser Example**

Сравнение DOM и SAX подходов.

**DOM**

* Парсер дерева моделей (на основе объекта) (Дерево узлов).
* DOM загружает файл в память и затем анализирует файл.
* Имеет ограничения памяти, так как он загружает весь XML файл перед разбором.
* DOM считывается и записывается (может вставлять или удалять узлы).
* Если содержимое XML невелико, то предпочитайте парсер DOM.
* Поиск назад и вперед возможен для поиска тегов и оценки информации внутри тегов. Таким образом, это упрощает навигацию.
* Медленнее во время выполнения.

**SAX**

* Парсер основан на событиях (последовательность событий).
* SAX анализирует файл по мере его чтения, т.е. анализирует node за node.
* Нет ограничений памяти, так как он не хранит содержимое XML в памяти.
* SAX доступен только для чтения, т.е. не может вставить или удалить node.
* Используйте синтаксический анализатор SAX, когда объем содержимого большой.
* SAX читает XML файл сверху вниз, а обратная навигация невозможна.
* Быстрее во время выполнения.

**Задача 4**. Достать всю информацию про всех сотрудников из файла xml\_file1.xml.

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

<company>

<name>IT-Heaven</name>

<offices>

<office floor="1" room="1">

<employees>

<employee name="Maksim" job="Middle Software Developer" />

<employee name="Ivan" job="Junior Software Developer" />

<employee name="Franklin" job="Junior Software Developer" />

</employees>

</office>

<office floor="1" room="2">

<employees>

<employee name="Herald" job="Middle Software Developer" />

<employee name="Adam" job="Middle Software Developer" />

<employee name="Leroy" job="Junior Software Developer" />

</employees>

</office>

</offices>

</company>

SAX-обработчик устроен так, что он считывает последовательно XML файлы и реагирует на разные события, после чего передает информацию специальному обработчику событий.

Часто используемые события:

1. startDocument — начало документа
2. endDocument — конец документа
3. startElement — открытие элемента
4. endElement — закрытие элемента
5. characters — текстовая информация внутри элементов.

Все события обрабатываются в обработчике событий, который нужно создать и переопределить методы.

Создание экземпляра класса javax.xml.parsers.SAXParser происходит на базе созданного экземпляра класса javax.xml.parsers.SAXParserFactory.

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

После создания парсера необходим обработчик его событий. Класс org.xml.sax.helpers.DefaultHandler является базовым классом для всех обработчиков событий. Но удобнее всего создать свой отдельный класс и переопределить в нем нужные методы.

**private** **static** **class** XMLHandler **extends** DefaultHandler {

@Override

**public** **void** startDocument() **throws** SAXException {

// Тут будет логика реакции на начало документа

}

@Override

**public** **void** endDocument() **throws** SAXException {

// Тут будет логика реакции на конец документа

}

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

// Тут будет логика реакции на начало элемента

}

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

// Тут будет логика реакции на конец элемента

}

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {

// Тут будет логика реакции на текст между элементами

}

@Override

**public** **void** ignorableWhitespace(**char**[] ch, **int** start, **int** length) **throws** SAXException {

// Тут будет логика реакции на пустое пространство внутри элементов (пробелы, переносы строчек и так далее).

}

}

Все события события SAX – методы обработчика событий выбрасывают SAXException.

После создания парсера и класса обработчика событий остается создать этот обработчик и вместе с XML-файлом отправить его парсеру.

DefaultHandler handler = new XMLHandler();

File file = new File("department-data.xml");

saxParser.parse(file, handler);

Замечание.

* Метод characters: если в элементе будет текст, например, «hello», то, теоретически, метод способен вызваться 5 раз подряд на каждый отдельный символ, однако это не страшно, так как все равно все будет работать.
* В методах startElement и endElement: uri – это пространство, в котором находится элемент, localName – это имя элемента с префиксом, qName – это просто имя элемента.
* uri и localName всегда пустые, если в фабрике не подключена обработка пространств. Для подключения используется метод фабрики setNamespaceAware(true). В этом случае можно получать пространство (uri) и элементы с префиксами перед ними (localName).

Решение.

1. Создать класс Employee с полями name, job, гетерами и конструктором.
2. Создать список всех сотрудников

**private** **static** ArrayList<Employee> employees = **new** ArrayList<>();

1. В данном случае нужная информация содержится в атрибутах элементов employee. У метода startElement есть параметр attributes. Поэтому в обработчике событий достаточно переопределить только один этот метод. Если имя элемента — employee, то получить информацию про его атрибуты. В attributes есть полезный метод, где, зная название атрибута, можно получить его значение. Именно его мы и использовали.

**private** **static** **class** XMLHandler **extends** DefaultHandler {

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

**if** (qName.equals("employee")) {

String name = attributes.getValue("name");

String job = attributes.getValue("job");

employees.add(**new** Employee(name, job));

}

}

}

1. Выполнить запрос XML файла.

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser parser = factory.newSAXParser();

XMLHandler handler = **new** XMLHandler();

parser.parse(**new** File("resource/xml\_file1.xml"), handler);

**for** (Employee employee : employees)

System.out.println(String.format("Имя сотрудника: %s, его должность: %s", employee.getName(), employee.getJob()));

**Задача 5**. Достать всю информацию обо всех сотрудниках из файла xml\_file2.xml.

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

<company>

<name>IT-Heaven</name>

<offices>

<office floor="1" room="1">

<employees>

<employee>

<name>Maksim</name>

<job>Middle Software Developer</job>

</employee>

<employee>

<name>Ivan</name>

<job>Junior Software Developer</job>

</employee>

<employee>

<name>Franklin</name>

<job>Junior Software Developer</job>

</employee>

</employees>

</office>

<office floor="1" room="2">

<employees>

<employee>

<name>Herald</name>

<job>Middle Software Developer</job>

</employee>

<employee>

<name>Adam</name>

<job>Middle Software Developer</job>

</employee>

<employee>

<name>Leroy</name>

<job>Junior Software Developer</job>

</employee>

</employees>

</office>

</offices>

</company>

Информация об имени и должности хранится в XML файле как текстовая внутри элементов name и job. Добавим соответствующие поля в класс-обработчик для хранения данных о сотруднике. А так же поле lastElementName для фиксирования текущего элемента.

**private** **static** **class** AdvancedXMLHandler **extends** DefaultHandler {

**private** String name, job, lastElementName;

}

В методе startElement сохраним имя текущего элемента.

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

lastElementName = qName;

}

Если там еще осталась информация, то, значит, это нужный нам текст, а далее мы определяем, имя это или профессия, используя lastElementName. В методе endElement мы проверяем, считана ли вся информация, и если считана, то мы создаем сотрудника и сбрасываем информацию.

Для считывания текста внутри элементов используется метод characters. Так как метод characters считывает все символы внутри элементов, в том числе все пробелы и переносы строчек, то нужно их игнорировать, то есть отфильтровать информацию. Если после фильтрации что-то останется, то это и будет нужный текст. Используя lastElementName можно определить, имя это или профессия. Сохраним это значение в соответствующий элемент name или job.

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {

String information = **new** String(ch, start, length);

information = information.replace("\n", "").trim();

**if** (!information.isEmpty()) {

**if** (lastElementName.equals("name"))

name = information;

**if** (lastElementName.equals("job"))

job = information;

}

}

В методе endElement проверяем, вся ли информация считана и завершаем создание элемента Employee, добавляем его в список.

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

**if** ( (name != null && !name.isEmpty()) && (job != null && !job.isEmpty()) ) {

employees.add(**new** Employee(name, job));

name = null;

job = null;

}

}

**Задача 6**. Дан элемент element, вывести имена и атрибуты всех внутренних элементов, если элемент не найден — вывести это. Информация хранится в файле xml\_file3.xml.

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

<root>

<oracle>

<connection value="jdbc:oracle:thin:@10.220.140.48:1521:test1" />

<user value="secretOracleUsername" />

<password value="111" />

</oracle>

<mysql>

<connection value="jdbc:mysql:thin:@10.220.140.48:1521:test1" />

<user value="secretMySQLUsername" />

<password value="222" />

</mysql>

</root>

Будем выводить всю информацию об элементах внутри root.

Класс обработчик должен знать значение искомого элемента. Будем передавать это значение в конструктор обработчика событий.

SearchingXMLHandler handler = **new** SearchingXMLHandler("root");

В классе опишем поле element для хранения переданного значения.

**private** **static** **class** SearchingXMLHandler **extends** DefaultHandler {

**private** String element;

**public** SearchingXMLHandler(String element) {

**this**.element = element;

}

}

Для того, чтобы знать, находимся ли мы внутри нужного элемента или нет, объявим в классе-обработчике логическую переменную isEntered.

**private** **boolean** isEntered;

При входе в искомый элемент нужно выставить флажок isEntered в true.

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

**if** (qName.equals(element)) {

isEntered = true;

}

}

Оказавшись внутри нужного элемента каждый новый дочерний элемент в startElement обрабатываем, зная, что он уже точно внутренний элемент искомого элемента. Таким образом, выводится имя элемента и его название.

**if** (isEntered) {

System.out.println(String.format("Найден элемент <%s>, его атрибуты:", qName));

**int** length = attributes.getLength();

**for**(**int** i = 0; i < length; i++)

System.out.println(String.format("Имя атрибута: %s, его значение: %s", attributes.getQName(i), attributes.getValue(i)));

}

При выходе из искомого элемента нужно выставить флажок isEntered в false.

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

**if** (qName.equals(element))

isEntered = false;

}

**Самостоятельная работа**

**Задание 1**. Создать вручную xml-файл вложенности не меньше 4-х о продаже квартир. Примерная структура файла может быть такой

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

<supply>

<region name="Ленинский">

<flat count="2" square="51.2" kitchen="14.6" housingpart="30.6">

<rooms>

<room>

19.0

</room>

<room>

11.6

</room>

</rooms>

<seller tel="+7922-154-10-11" title="Иванов Иван"/>

</flat>

</region>

<region name="Октябрьский">

<flat count="1" square="36.2" kitchen="9.9" housingpart="18.6">

<rooms>

<room>

18.6

</room>

</rooms>

<seller tel="+7922-154-10-11" title="Петров Петр"/>

</flat>

</region>

</supply>

Используя каждый из подходов, решите задачи:

1. Считайте данные обо всех квартирах.

2. Считайте данные обо всех квартирах какого-то района, название района вводится с клавиатуры.

**Задание 2**. Создать программным образом xml-файл. В качестве данных для файла можно взять данные из задания 1, предварительно изменив их. Вариант изменения – группировать не по районам, а по количеству комнат в квартирах. Название района можно сделать атрибутом квартиры.

Запись созданного элемента класса org.w3c.dom.Document в xml-файл.

**private static void** writeDocument(Document document) **throws** TransformerFactoryConfigurationError {

**try** {

//объект класса Transformer может использоваться для записи результатов преобразования в различные приемники с помощью метода transform

Transformer tr = TransformerFactory

.newInstance()

.newTransformer();

//Объекты класса DOMSource используются для хранения исходного дерева преобразования в форме дерева объектной модели документа (DOM).

DOMSource source = new DOMSource(document);

FileOutputStream fos = new

FileOutputStream("other.xml");

StreamResult result = new StreamResult(fos);

// метод transform(Source xmlSource, Result outputTarget) преобразует XML ресурс в какой-то из классов-наследников класса Result.

tr.transform(source, result);

} **catch** (TransformerException | IOException e) {

e.printStackTrace(System.out);

}

}

Наследники класса Result: DOMResult, JAXBResult, SAAJResult, SAXResult, StAXResult, StreamResult