# Лабораторная работа №5

# Хранение и обработка данных в Java.

*Цель работы:* изучение основ работы с файлами xml, properties и базами данных в java.

## Основные понятия XML

XML – это описанная в текстовом формате иерархическая структура, предназначенная для хранения любых данных. Визуально структура может быть представлена как дерево элементов. Элементы XML описываются тегами и могут иметь атрибуты.

Формат XML используется для хранения данных и обмена ими между несколькими информационными системами.

Первая строка XML-документа называется объявление XML – это строка, указывающая версию XML. В версии 1.0 объявление XML может быть опущено, в версии 1.1 оно обязательно. Также здесь может быть указана кодировка символов и наличие внешних зависимостей.

<?xml version="1.1" encoding="UTF-8"?>

Важнейшее обязательное синтаксическое требование заключается в том, что документ имеет только один корневой элемент.

Существуют две основные стратегии обработки XML документов: DOM (Document Object Model) и SAX (Simple API for XML). Основное их отличие связано с тем, что использование DOM позволяет читать и вносить изменения в существующий XML-документ, а также создавать новый. Стратегия использования SAX основывается на том, что содержимое XML-документа считывается, вызывая при этом наступление определенных событий. DOM должен весь документ считать, проанализировать и сохранить в памяти, а SAX-парсер обрабатывает XML-документ последовательно и не требует дополнительной памяти.

Для работы с XML-файлами Java располагает достаточно большим набором инструментов, начиная от встроенных возможностей, которые предоставляет Core Java, и заканчивая большим набором разнообразного стороннего кода, оформленного в виде библиотек.

## Объектная Модель Документа (DOM)

Объектная Модель Документа, называемая DOM, определяет набор интерфейсов для разобранной версии XML-документа. Парсер читает весь документ и строит дерево объектов в памяти, так что приложение Java может использовать интерфейсы DOM для манипулирования деревом. Предоставляются возможности двигаться по дереву, чтобы увидеть, что содержал исходный документ, удаления части дерева, перемещения элементов дерева, добавления новых ветвей и т.д. Технология DOM была создана организацией W3C.

### Проблемы DOM

DOM обеспечивает богатый набор функций, которые разработчик может использовать для интерпретации XML-документа и манипулирования им, но также имеются и недостатки:

* DOM строит в памяти дерево всего документа. Если документ очень большой, это требует значительного объема памяти.
* DOM создает объекты, которые представляют все, что есть в исходном документе, включая элементы, текст, атрибуты и пропуски. Если необходима только небольшая часть исходного документа, то крайне нерационально создавать все эти объекты, которые никогда не будут использованы.
* Парсер DOM должен прочитать весь документ прежде, чем исходный код получит управление. Для очень больших документов это может привести к значительной задержке.

Несмотря на это, API DOM является очень удобным способом для разбора XML-документов.

### Чтение документа при помощи DOM

Для чтения различных документов в Java имеется пакет javax.xml.parsers, который содержит различные классы, реализующие чтение и разбор файлов по определенным технологиям. Для чтения XML-документов в DOM используются классы DocumentBuilderFactory (позволяет получить парсер, порождающий дерево DOM объектов XML-документа) и DocumentBuilder (разбирает XML-документ и создает соответствующий DOM Document).

File xmlFile = **new** File("Path\_to\_file");

DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder = factory.newDocumentBuilder();

Document document = builder.parse(xmlFile);

Для хранения дерева узлов используется пакет org.w3c.dom. Ниже перечислены его основные классы:

|  |  |
| --- | --- |
| **Класс** | **Описание** |
| [Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) | Представляет атрибут в объектe Element. |
| Document | Представляет весь HTML или XML-документ. |
| Element | Представляет элемент (обычно тег) в HTML или XML-документе. |
| [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) | Представляет собой узел и является основным типом данных для всей объектной модели документа. В качестве узла могут выступать Attr, Document, Element, Text и некоторые другие. По сути, класс Node является родительским для вышеперечисленных. |
| [NodeList](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html) | Обеспечивает абстракцию упорядоченного списка узлов, не определяя или ограничивая, как этот список реализуется. |
| [Text](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Text.html) | Представляет текстовый контент объектов Element или Attr. |

Основные методы класса Attr:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getName](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html#getName%28%29)() | Возвращает имя этого атрибута. |
| [Element](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html) [getOwnerElement](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html#getOwnerElement%28%29)() | Возвращает узел, которому принадлежит данный атрибут. |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getValue](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html#getValue%28%29)() | Возвращает значение атрибута. |
| void [setValue](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html#setValue%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) value) | Устанавливает значение атрибута. |

Основные методы класса Document:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| [Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) [createAttribute](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#createAttribute%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Создает атрибут с указанным именем. |
| [Element](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html) [createElement](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#createElement%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) tagName) | Создает элемент с указанным именем. |
| [Text](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Text.html) [createTextNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#createTextNode%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) data) | Создает узел Text с указанным значением. |
| [Element](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html) [getDocumentElement](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#getDocumentElement%28%29)() | Возвращает корневой элемент документа. |
| [Element](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html) [getElementById](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#getElementById%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) elementId) | Возвращает элемент по его ID. |
| [NodeList](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html) [getElementsByTagName](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#getElementsByTagName%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) tagname) | Возвращает список элементов документа с данным именем тега. |
| [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) [importNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#importNode%28org.w3c.dom.Node,%20boolean%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) importedNode, boolean deep) | Импортирует узел от другого документа. |
| void [normalizeDocument](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#normalizeDocument%28%29)() | Этот метод действует, как будто документ проходил через сохранение и цикл загрузки, помещая документ в "нормальную" форму. |
| [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) [renameNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Document.html#renameNode%28org.w3c.dom.Node,%20java.lang.String,%20java.lang.String%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) n, [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) namespaceURI, [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) qualifiedName) | Переименование атрибута или элемента. |

Основные методы класса Element:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getAttribute](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#getAttribute%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Получает значение атрибута по имени. |
| [Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) [getAttributeNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#getAttributeNode%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Получает узел атрибута по имени. |
| [NodeList](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html) [getElementsByTagName](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#getElementsByTagName%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Возвращает список потомков элемента с данным именем тега. |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getTagName](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#getTagName%28%29)() | Возвращает имя элемента. |
| boolean [hasAttribute](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#hasAttribute%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Возвращает true когда атрибут с именем имеется у этого элемента, false иначе. |
| void [removeAttribute](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#removeAttribute%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name) | Удаляет атрибут по имени. |
| [Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) [removeAttributeNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#removeAttributeNode%28org.w3c.dom.Attr%29)([Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) oldAttr) | Удаляет указанный узел атрибута. |
| void [setAttribute](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#setAttribute%28java.lang.String,%20java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) name, [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) value) | Добавляет новый атрибут. |
| [Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) [setAttributeNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Element.html#setAttributeNode%28org.w3c.dom.Attr%29)([Attr](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Attr.html) newAttr) | Добавляет новый узел атрибута. |

Основные поля класса Node:

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| static short [ATTRIBUTE\_NODE](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#ATTRIBUTE_NODE) | Узел является атрибутом |
| static short [DOCUMENT\_NODE](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#DOCUMENT_NODE) | Узел является документом. |
| static short [ELEMENT\_NODE](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#ELEMENT_NODE) | Узел является элементом. |
| static short [TEXT\_NODE](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#TEXT_NODE) | Узел является текстом. |

Основные методы класса Node:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Node [appendChild](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#appendChild%28org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) newChild) | Добавляет узел newChild в конец списка дочерних элементов этого узла. |
| Node [cloneNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#cloneNode%28boolean%29)(boolean deep) | Возвращает копию этого узла. |
| [NamedNodeMap](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NamedNodeMap.html) [getAttributes](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getAttributes%28%29)() | Возвращает именованную карту атрибутов этого узла (если это Element) или null иначе. |
| [NodeList](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html) [getChildNodes](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getChildNodes%28%29)() | Возвращает все дочерние элементы этого узла. |
| Node [getFirstChild](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getFirstChild%28%29)() | Возвращает первый дочерний элемент этого узла. |
| Node [getLastChild](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getLastChild%28%29)() | Возвращает последний дочерний элемент этого узла. |
| Node [getNextSibling](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getNextSibling%28%29)() | Возвращает следующий узел. |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getNodeName](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getNodeName%28%29)() | Возвращает имя этого узла, в зависимости от его типа. |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getNodeValue](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getNodeValue%28%29)() | Возвращает значение этого узла, в зависимости от его типа. |
| Node [getParentNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getParentNode%28%29)() | Возвращает родительский узел. |
| Node [getPreviousSibling](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getPreviousSibling%28%29)() | Возвращает предыдущий узел. |
| [String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) [getTextContent](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#getTextContent%28%29)() | Возвращает текстовый контент этого узла и его потомков. |
| boolean [hasAttributes](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#hasAttributes%28%29)() | Возвращает true, если у этого узла (если это - элемент) есть какие-либо атрибуты. |
| boolean [hasChildNodes](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#hasChildNodes%28%29)() | Возвращает true, если у этого узла (если это - элемент) есть какие-либо дочерние элементы. |
| Node [insertBefore](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#insertBefore%28org.w3c.dom.Node,%20org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) newChild, [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) refChild) | Вставляет узел newChild перед существующим дочерним узлом refChild. |
| boolean [isEqualNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#isEqualNode%28org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) arg) | Сравнивает два узла. |
| boolean [isSameNode](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#isSameNode%28org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) other) | Возвращает true, если этот узел является тем же самым узлом как other. |
| Node [removeChild](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#removeChild%28org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) oldChild) | Удаляет дочерний узел oldChild из списка дочерних элементов, и возвращает его. |
| Node [replaceChild](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#replaceChild%28org.w3c.dom.Node,%20org.w3c.dom.Node%29)([Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) newChild, [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) oldChild) | Заменяет дочерний узел oldChild на newChild в списке дочерних элементов, и возвращает oldChild |
| void [setNodeValue](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#setNodeValue%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) nodeValue) | Устанавливает значение этого узла, в зависимости от его типа. |
| void [setTextContent](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html#setTextContent%28java.lang.String%29)([String](http://spec-zone.ru/RU/Java/Docs/7/api/java/lang/String.html) textContent) | Устанавливает текстовый контент этого узла и его потомков. |

Основные методы класса NodeList:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| int [getLength](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html#getLength%28%29)() | Число узлов в списке. |
| [Node](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/Node.html) [item](http://spec-zone.ru/RU/Java/Docs/7/api/org/w3c/dom/NodeList.html#item%28int%29)(int index) | Возвращает из набора узел с индексом index. |

Следует отметить, что т.к. классы Attr, Document, Element и Text являются наследниками от Node, то они также унаследовали и все его методы.

После получения объекта Document от парсера вся дальнейшая работа основывается на использовании объектов пакета org.w3c.dom в зависимости от задач, решаемых приложением.

Пример файла XML:

<?xml version="1.0"?>

<Languages>

<Language>

<name>Java</name>

<age>21</age>

</Language>

<Language>

<name>C</name>

<age>44</age>

</Language>

</Languages>

Пример исходного кода парсера:

**import** java.io.File;

**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.Element;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**public** **class** ReadDOM {

**public** **static** **void** main(String[] args) {

String filepath = "./bin/languages.xml";

File xmlFile = **new** File(filepath);

DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder;

**try** {

builder = factory.newDocumentBuilder();

Document document = builder.parse(xmlFile);

// теперь XML полностью загружен в память

document.getDocumentElement().normalize();

System.***out***.println("Корневой элемент: " + document.getDocumentElement().getNodeName());

// получаем узлы с именем Language

NodeList nodeList = document.getElementsByTagName("Language");

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

Node node = nodeList.item(i);

// если узел является элементом

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

Element element = (Element) node;

System.***out***.println("\nЭлемент "+element.getNodeName());

// получаем список дочерних

NodeList childNodesList = element.getChildNodes();

**for** (**int** j = 0; j < childNodesList.getLength(); j++) {

Node childNode = childNodesList.item(j);

// если узел является элементом

**if** (childNode.getNodeType() == Node.***ELEMENT\_NODE***) {

System.***out***.println(childNode.getNodeName()+": "+ childNode.getTextContent());

}

}

}

}

} **catch** (Exception exc) {

exc.printStackTrace();

}

}

}

### Запись документа при помощи DOM

Для записи XML-документа потребуется создание пустого объекта Document. Для этого необходимо использовать у объекта DocumentBuilder не метод parse(), а метод newDocument().

Далее при помощи вышеописанных методов необходимо создать корректное дерево узлов и записать его в файл или консоль. Для этого необходимо использовать ряд классов из пакета javax.xml.transform, таких как:

* TransformerFactory – конструктор для преобразований.
* Transformer – объект для выполнения фактического преобразования.
* OutputKeys – позволяет задать параметры форматирования документа.
* dom.DOMSource – получает исходный код созданного документа.
* stream.StreamResult – создает объект для записи в него документа.

Пример:

**import** java.io.File;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.transform.OutputKeys;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.dom.DOMSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.Node;

**public** **class** WriteDOM {

**public** **static** **void** main(String[] args) {

DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder;

**try** {

builder = factory.newDocumentBuilder();

Document doc = builder.newDocument();// создаем пустой объект Document

// создаем корневой элемент

Element rootElement = doc.createElement("Languages");

// добавляем корневой элемент в объект Document

doc.appendChild(rootElement);

// добавляем первый дочерний элемент к корневому

rootElement.appendChild(*getLanguage*(doc, "1", "Java", "21"));

//добавляем второй дочерний элемент к корневому

rootElement.appendChild(*getLanguage*(doc, "2", "C", "44"));

doc.getDocumentElement().normalize();

//создаем объект TransformerFactory для преобразования документа в файл

TransformerFactory transformerFactory = TransformerFactory.*newInstance*();

Transformer transformer = transformerFactory.newTransformer();

// установка параметров форматирования для красивого вывода

transformer.setOutputProperty(OutputKeys.***INDENT***, "yes");

transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

//получение исходного кода готового документа

DOMSource source = **new** DOMSource(doc);

//создание объекта для записи - файл

StreamResult file = **new** StreamResult(**new** File("./bin/languages\_new.xml"));

//запись данных

transformer.transform(source, file);

System.***out***.println("Создание XML файла закончено");

} **catch** (Exception e) {

e.printStackTrace();

}

}

// метод для создания нового узла XML-файла

**private** **static** Node getLanguage(Document doc, String id, String name, String age) {

Element language = doc.createElement("Language");

language.setAttribute("id", id); // устанавливаем атрибут id

// создаем элементы name и age

language.appendChild(*getLanguageElements*(doc, language, "name", name));

language.appendChild(*getLanguageElements*(doc, language, "age", age));

**return** language;

}

// метод для создания одного узла

**private** **static** Node getLanguageElements(Document doc, Element element, String name, String value) {

Element node = doc.createElement(name);

node.appendChild(doc.createTextNode(value));

**return** node;

}

}

### Изменение документа при помощи DOM

Редактирование XML-документа сводится к созданию объекта Document из существующего файла (чтение документа), преобразованию его узлов рассмотренными выше методами и запись измененного документа в тот же самый файл.

## Simple API for XML

Одной из альтернатив DOM является Simple API for XML, или SAX. SAX позволяет обрабатывать документ по мере его чтения, что устраняет необходимость ожидать выполнения каких-то действий, пока весь документ не будет сохранен.

SAX имеет несколько характеристик, направленных на преодоление недостатков DOM:

* Парсер SAX посылает в код события(вместе с их параметрами). Разработчик самостоятельно решает, какие события для него важны и какой вид структуры данных он хочет создать для хранения данных из этих событий. Если эти данные явным образом не сохраняются, то они теряются.
* Парсер SAX не создает никаких объектов вообще, он просто доставляет события в приложение. Если необходимо создавать какие-либо объекты на основе этих событий – это необходимо сделать явным образом вручную.
* Парсер SAX начинает генерировать события сразу же после начала своей работы. Программный код получает событие, когда парсер найдет начало документа, когда он найдет начало элемента, когда он найдет текст и т.д. Приложение сразу же начнет генерировать результаты и не будет ожидать, пока будет разобран весь документ. Чтобы остановить парсер SAX необходимо сгенерировать соответствующее исключение.

### Проблемы SAX

Парсеры SAX также имеют проблемы:

* События SAX не сохраняют состояния. Когда парсер SAX находит текст в XML-документе, он посылает событие в программный код. Это событие просто передает найденный текст, оно не сообщает, какой именно элемент содержит этот текст. Если необходимо это знать, то разработчик должен написать код, управляющий состоянием, самостоятельно.
* События SAX не сохраняются. Если приложению нужна структура данных, которая моделирует XML-документ, этот код необходимо написать самостоятельно. Т.е. если данные не были где-либо сохранены, то они теряются и для повторного обращения к ним необходимо снова парсить весь XML-документ.
* SAX не управляется централизованной организацией, например, такой как W3C.

### Как работает обработка в SAX

SAX анализирует поток XML и проходит через него. Рассмотрим следующий XML-код:

<?xml version="1.0"?>

<Root>

<tag1>Текст1</tag1>

<tag2>Text2</tag2>

</Root>

Процессор SAX, анализирующий этот код, будет генерировать, как правило, следующие события:

Начало документа

Начало элемента (Root)

Символы (пропуск)

Начало элемента (tag1)

Символы (Текст1)

Конец элемента (tag1)

Символы (пропуск)

Начало элемента (tag2)

Символы (Text2)

Конец элемента (tag2)

Символы (пропуск)

Конец элемента (Root)

Конец документа

SAX API дает возможность разработчику поймать эти события и работать по ним.

Обработка в SAX включает в себя следующие шаги:

1. Создание обработчика событий.
2. Создание парсера SAX.
3. Назначение обработчика событий для парсера.
4. Разбор документа с посылкой каждого события в обработчик.

### Чтение документа при помощи SAX

Чтение документа XML сводится к переопределению методов, соответствующих событиям, рассмотренным в предыдущем разделе. Для этого используются классы пакета пакет javax.xml.parsers, такие как SAXParserFactory (позволяет получить парсер SAX) и SAXParser (разбирает XML-документ и генерирует события).

Для непосредственной обработки данных используется пакет org.xml.sax и такие его классы как:

* helpers.DefaultHandler – класс, содержащий методы, соответствующие событиям SAX, которые необходимо переопределить.
* Attributes – позволяет работать с атрибутами при наступлении события начала элемента.
* SAXException – генерируемое исключение в случае невозможности разобрать XML-документ.

Пример:

**import** javax.xml.parsers.SAXParser;

**import** javax.xml.parsers.SAXParserFactory;

**import** org.xml.sax.Attributes;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** SAXReader {

**public** **static** **void** main(String args[]) {

String filepath = "./bin/languages.xml";

**try** {

SAXParserFactory factory = SAXParserFactory.*newInstance*();

SAXParser saxParser = factory.newSAXParser();

// анонимный класс, расширяющий класс DefaultHandler

DefaultHandler handler = **new** DefaultHandler() {

String tag = ""; // Строка для хранения имени текущего тега

// Метод вызывается когда SAXParser генерирует начало тега

@Override

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

**if** (qName.equalsIgnoreCase("Language"))

System.***out***.println("\nЭлемент "+qName);

tag = qName;

}

// Метод вызывается когда SAXParser считывает текст между тегами

@Override

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

**if** (tag.equalsIgnoreCase("name"))

System.***out***.println("name: " + **new** String(ch, start, length));

**else**

**if** (tag.equalsIgnoreCase("age"))

System.***out***.println("age: " + **new** String(ch, start, length));

}

// Метод вызывается когда SAXParser генерирует конец тега

@Override

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

tag = "";

}

};

// Старт разбора методом parse, которому передается наследник от DefaultHandler

saxParser.parse(filepath, handler);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

## Файлы \*.properties

Файлы свойств предназначены для того, чтобы хранить в них какие-то статические данные, необходимые для проекта, например конфигурационные параметры программы.

Данные в этих файлах представлены в виде «**ключ =|: значение**», где

ключ – это уникальное имя, по которому можно получить доступ к значению, хранимому под этим ключом.

значение – это текст, либо число, которое вам необходимо для выполнения определённой логики в программе.

Разделителем могут выступать знаки «=» или «:».

\*.properties файлы могут использовать знак решетки (#) или восклицательный знак (!) как первый, не пустой символ в строке для обозначения последующего текста в качестве комментария.

Для чтения и разбора файла \*.properties используется объект класса java.util.Properties. Метод load() считывает файл, а метод getProperty("свойство") возвращает значение указанного свойства.

Пример файла \*.properties (сохранён в кодировке UTF-8):

lab : Лабораторная работа №5

fio : Веретенников Олег Владимирович

Пример считывания файла \*.properties:

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.util.Properties;

**public** **class** Prop {

**public** **static** **void** main(String[] args) {

Properties prop = **new** Properties();

**try** {

//обращение к файлу и получение данных

FileInputStream fis = **new** FileInputStream("./resourсes/config.properties");

prop.load(fis);

// взятие свойства и преобразование в необходимую кодировку

String lab = **new** String(prop.getProperty("lab").getBytes("ISO8859-1"));

String fio = **new** String(prop.getProperty("fio").getBytes("ISO8859-1"));

//печать полученных данных в консоль

System.***out***.println("lab: " + lab + "\nfio: " + fio);

} **catch** (IOException e) {

System.***out***.println("Ошибка в программе: файл не найден");

e.printStackTrace();

}

}

}

## Работа с базой данных

Для того, чтобы получить доступ к базе данных, Java позволяет использовать JDBC (Java Database Connectivity) API, который входит в стандартную библиотеку Java. JDBC позволяет подключиться к любой базе данных: Postgres, MySQL, SQL Server, Oracle и т. д. – при наличии соответствующей реализации драйвера, необходимого для подключения. JDBC – это стандарт взаимодействия приложения с различными СУБД. Он основан на концепции драйверов, позволяющей получать соединение с БД по специальному URL.  
Из определения выходит, что для соединения с СУБД нужен драйвер, который можно скачать сайте производителя СУБД.

Далее будет рассматриваться работа с СУБД MySQL. Для скачивания драйвера необходимо перейти по адресу https://dev.mysql.com/downloads/connector/j/ и сохранить архив. Потом нужно скопировать библиотеку JAR в проект и указать путь к ней в настройках. JAR-файл содержит класс com.mysql.jdbc.Driver, необходимый для подключения к MySQL. После этого необходимо выполнить строку Class.*forName*("com.mysql.jdbc.Driver"), которая загружает класс и этим гарантирует регистрацию драйвера в DriverManager.

Все основные сущности в JDBC API, с которыми наиболее часто приходится работать, являются интерфейсами:

* Connection.
* Statement.
* PreparedStatement.
* CallableStatement.
* ResultSet.
* DatabaseMetaData.

JDBC драйвер конкретной СУБД как раз и предоставляет реализации этих интерфейсов.

### Connection

Используется для соединения с сервером баз данных.

Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:port/dbName", "login", "password");

### Statement

Данный объект используется для выполнения простых SQL-запросов без параметров. Содержит базовые методы для выполнения запросов и извлечения результатов.

Statement stmt = *con*.createStatement();

Класс Statement имеет три различных метода выполнения SQL-выражений: executeQuery, executeUpdate и execute, которые вызываются в зависимости от текста SQL-запроса.

Метод **executeQuery** используется в запросах, результатом которых является один единственный набор значений, таких как запросов типа SELECT. Возвращает объект ResultSet, содержащий результат запроса.

Метод **executeUpdate** следует использовать, как для выполнения операторов управления данными типа INSERT, UPDATE или DELETE (DML - Data Manipulation Language), так и для операторов определения структуры базы данных CREATE TABLE, DROP TABLE (DDL - Data Definition Language).

Результатом выполнения операторов INSERT, UPDATE, или DELETE является изменения одной или более строк таблицы. Метод executeUpdate возвращает целочисленное значение, определяющее, сколько строк было модифицировано. Для выражений DML, которые не оперируют со строками, возвращаемое методом executeUpdate значение всегда равно нулю.

Метод **execute** используется, когда операторы SQL возвращают более одного набора данных, более одного счетчика обновлений или и то, и другое. Такая возможность редко используется.

Все эти методы принимают один параметр – текст SQL-запроса.

Объекты Statement закрываются автоматически сборщиком мусора виртуальной машины Java. Тем не менее рекомендуется закрывать их явно после того, как работа с ними завершена. Закрытие объектов Statement после их использования освобождает ресурсы СУБД и позволяет избежать проблем с памятью.

Объект Statement считается завершенным, если он выполнился и все его результаты были возвращены. Для метода executeQuery, возвращающий набор данных, оператор считается завершенным, если считаны все строки соответствующего объекта ResultSet. В случае использования метода executeUpdate объект завершен сразу же после выполнения метода. В случае вызова метода execute оператор остается не завершенным до тех пор, пока все наборы данных не будут считаны.

### PreparedStatement

Является наследником от Statement и используется для выполнения SQL-запросов с или без входных параметров. Добавляет методы управления входными параметрами. Также часто используется для выполнения множества похожих запросов.

Необходимо отметить, что PreparedStatement, наследующий все методы Statement, имеет свои реализации методов executeQuery, executeUpdate и execute. Объекты PreparedStatement не принимают SQL-выражения в виде аргументов этих методов, так как они уже содержат прекомпилированные SQL-выражения.

Например:

PreparedStatement st = *con*.prepareStatement("insert into user(id,name) values(?,?)");

st.setInt(1, 3);

st.setString(2, "myname");

st.executeUpdate();

PreparedStatement поддерживает пакетную отправку SQL запросов, что значительно уменьшает траффик между клиентом и базой данных. Пример:

PreparedStatement st = *con*.prepareStatement("insert into user(id,name) values(?,?)");

st.setInt(1, 4);

st.setString(2, "name1");

st.addBatch();

st.setInt(1, 5);

st.setString(2, "name2");

st.addBatch();

st.executeBatch();

Методы **setТипДанных** позволяют установить любой из параметров запроса, описанных при помощи знака «?» в SQL-выражении. Номер параметра задается первым аргументом, а значение вторым.

### CallableStatement

Является наследником от PreparedStatement и используется для вызовов хранимых процедур. Добавляет методы для манипуляции выходными параметрами.

CallableStatement наследуют методы от PreparedStatement без параметров. Использование аргументов в методах executeXXX объектов PreparedStatement и CallableStatement приведет к генерации ошибки SQLException.

### ResultSet

Представляет результирующий набор данных и обеспечивает приложению построчный доступ к результатам запросов. При обработке запроса ResultSet поддерживает указатель на текущую обрабатываемую запись.

Доступ к данным ResultSet обеспечивает посредством набора get-методов, которые организуют доступ к колонкам текущей строки. Метод ResultSet.next используется для перемещения к следующей записи ResultSet, делая ее текущей.

Список наиболее часто используемых методов класса ResultSet представлен в таблице:

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| public boolean absolute(int row) | Метод перемещает курсор на заданное число записей от начала, если число положительно, и от конца - если отрицательно |
| public void afterLast() | Этот метод перемещает курсор в конец результирующего набора за последнюю запись |
| public void beforeFirst() | Этот метод перемещает курсор в начало результирующего набора перед первой записью |
| public void deleteRow() | Удаляет текущую запись из результирующего набора |
| public ResultSetMetaData getMetaData() | Предоставляет объект метаданных для данного ResultSet. Класс ResultSetMetaData содержит информацию о результирующие таблице, такую как количество столбцов, их заголовок и т.д. |
| public int getRow() | Возвращает номер текущей записи |
| public ТипДанных getТипДанных(Поле) | Группа методов, которые возвращают значение поля указанного типа данных. В качестве параметра могут принимать номер или имя поля. |
| public Statement getStatement() | Возвращает экземпляр Statement, который произвел данный результирующий набор |
| public boolean next()  public boolean previous() | Эти методы позволяют переместиться в результирующем наборе на одну запись вперед или назад. Во вновь созданном результирующем наборе курсор устанавливается перед первой записью, поэтому первое обращение к методу next() влечет позиционирование на первую запись. Эти методы возвращают true, если остается запись для дальнейшего перемещения. Если записей для обработки больше нет, возвращается false. |
| public void close() | Осуществляет немедленное закрытие ResultSet вручную. |

### DatabaseMetaData

Позволяет получить метаинформацию о схеме базы данных, а именно какие в базе данных есть объекты - таблицы, колонки, индексы, триггеры, процедуры и так далее.

### Транзакции в JDBC

По умолчанию каждое SQL-выражение автоматически исполняется при выполнении statement.execute и подобных методов. Для того, чтобы открыть транзакцию сначала необходимо установить флаг autoCommit у соединения в значение false. Далее необходимо использовать методы commit для выполнения транзакции и rollback для её отката.

*con*.setAutoCommit(**false**);

Statement st = *con*.createStatement();

**try** {

st.execute("insert into user(name) values('name3')");

*con*.commit();

} **catch** (SQLException e) {

*con*.rollback();

}

Пример работы с базой данных:

**import** java.sql.\*;

**public** **class** DataBase {

**private** **static** Connection *con*;

**private** **static** Statement *stmt*;

**private** **static** ResultSet *rs*;

**public** **static** **void** main(String args[]) {

**try** {

// регистрация драйвера

Class.*forName*("com.mysql.jdbc.Driver");

**try** {

// соединение с сервером

*con* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/test", "root", "");

// создание объекта Statement

*stmt* = *con*.createStatement();

// Выполнение запроса

*rs* = *stmt*.executeQuery("SELECT \* FROM table1");

// Обход всех записей

**while** (*rs*.next()) {

String f1 = *rs*.getString("f1");

String f2 = *rs*.getString("f2");

System.***out***.println("field1 : " + f1 + " field2 : " + f2 + "\n");

}

} **catch** (SQLException sqlEx) {

sqlEx.printStackTrace();

} **finally** {

// закрытие соединения

**try** { *con*.close(); } **catch**(SQLException se) {}

**try** { *stmt*.close(); } **catch**(SQLException se) { }

**try** { *rs*.close(); } **catch**(SQLException se) { }

}

} **catch** (ClassNotFoundException e) {

System.***out***.println("Отсуствует драйвер");

e.printStackTrace();

**return**;

}

}

}

## Задание к лабораторной работе:

1. Разработать формат представления данных в xml и БД. В xml предусмотреть все рассмотренные виды узлов.
2. Реализовать методы: добавление, удаление и редактирование записей в каждом формате представления. Пользователю необходимо предоставить выбор формата представления, в котором будут производиться изменения.
3. Реализовать возможность конвертирования данных из xml в БД и из БД в xml.
4. Для чтения XML-документа использовать технологию SAX, для записи DOM.
5. Реализовать поиск по заданным параметрам. Поиск может осуществляться как по одному, так и по нескольким параметрам. Если параметр числовой, то должна быть возможность поиска максимальных и минимальных значений.
6. Минимальное количество записей в БД или xml 10.
7. В файле \*.properties необходимо хранить:
   * Настройки подключения к БД;
   * Рабочий каталог;
   * Ограничения связанные с заданием (например минимальная оценка 1 максимальная 5 и т.д.)
   * Тексты информационных сообщений и сообщений об ошибках

Варианты заданий:

|  |  |
| --- | --- |
| **№** | **Задание** |
| 1 | Лекарственные средства  (Название, тип, стоимость, состав, противопоказания и т.д.). |
| 2 | Список учащихся (Школа, класс, фамилия, имя, отчество, оценки и т.д.) |
| 3 | Страны (Континент, название, площадь, численность, полезные ископаемые, тип власти и т.д.) |
| 4 | Коллекция музыкальных произведений (название, автор, тип, ссылка для скачивания и т.д.) |
| 5 | Города (Страна, название, мэр города, численность, площадь, уровень жизни и т.д.) |
| 6 | Школы (Регион, город или населенный пункт, адрес, название, ФИО директора и т.д.) |
| 7 | Каталог сайтов (Название, url, описание, ключевые слова, категория сайта и т.д.) |
| 8 | Общественные места (стадионы, больницы, пляжи и т.д. )  (Название города, название места, описание, gps - координаты, тип и т.д.). |
| 9 | Автомобили(Марка, модель, цвет, гос. номер, фамилия, имя, отчество владельца и т.д.) |
| 10 | Список событий (Название, тип, дата и время начала, дата и время окончания, организатор, место проведения и т.д.) |

## Каждый отчет должен содержать:

1. Заголовок лабораторной работы (название и цель работы).
2. Фамилия, инициалы и группа студента.
3. Задание к лабораторной работе.
4. Краткие теоретические сведения.
5. Описание алгоритмов, функций, примененных решений.
6. Результаты выполнения программ.
7. Исходный код программ.
8. Выводы о проделанной работе.