Факультет комп’ютерних наук

Залікова робота

«*Крос-платформне програмування*»

**Варіант №2**

П.І.Б. Безрук Юрій Русланович

Група КС-21

1. Особливості SAX та DOM парсерів. Основи створення та використання SAX, DOM парсерів. (*10 балів*)

SAX и DOM парсеры используются синтаксического анализа и изменения XML-документов. SAX-парсеры являются парсерами, основанными на событиях. Они последовательно просматривают документ, анализируют его, отмечая события, которыми считаются появления очередных элементов XML, таких как открывающий или закрывающий тег или текст, содержащийся внутри элемента. Когда парсер встречает элемент, он вызывает соответствующий метод обработки, предназначенный для этого типа элемента. Эти методы прописываются в классе-обработчике, который является наследником класса DefaultHandler. Наиболее часто используемые методы-обработчки:

public void startDocument() –вызывается при начале обработки документа, в этом методе задаются какие-либо начальные действия, которые нужно выполнить при старте обработки документа (создание каких либо структур, инициализация переменных и т.д.)

public void endDocument() – вызывается при обработке конечного тега документа.

public void startElement(String uri, String name, String qname Attributes attrs ) – вызывается парсером , когда тот встречает символ «<», тоесть при открывающем теге элемента. В метод передается идентификатор простанства имен uri, локальное имя тега без префикса name, расшренное имя с префиксом qname и атрибуты элемента attrs, если они есть. Первые два аргумента равны null, если если пространство имен не определено.

public void endElement(String uri, String name, String qname) – вызывается при появлении символов «</», тоесть при появлении закрывающего тега элемента.

public void characters(char[] ch, int start, int length) – вызывается когда парсер встречает текстовое содержимое. В метод передается массив символов ch, индекс начала строки start и количество символов length.

Пример обработчика:

**public class NotebookHandler extends DefaultHandler{**

**@Override**

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

**System.*out*.println("Start document processing...");**

**}**

**@Override**

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

**System.*out*.println("Stop document processing.");**

**}**

**@Override**

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

**System.*out*.println("Start element"+qName+"processing...");**

**}**

**@Override**

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

**System.*out*.println("Stop element"+qName+"processing...");**

**}**

**@Override**

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

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

**System.*out*.print(str.trim());**

**}**

**}**

После того, как обработчик написан, нужно инициализировать парсинг документа. Для этого вначале необходимо создать фабрику парсера, с помощнее создать парсер, затем создать объект класса-обработчика и входной файловый поток с XML-документом. После чего у парсера вызывается метод parse(), куда передаются обработчик и поток с файлом.

Пример приведен ниже:

**SAXParserFactory factory = SAXParserFactory.*newInstance*();**

**try {**

**SAXParser parser = factory.newSAXParser();**

**DefaultHandler handler = new NotebookHandler();**

**InputStream xmlStream = new FileInputStream("data.xml");**

**parser.parse(xmlStream, handler);**

**} catch (ParserConfigurationException | SAXException | IOException e) {**

**e.printStackTrace();**

**}**

DOM-парсеры производят анализ, основываясь на структуре дерева, отражающего вложенность элементов документа. Из вложенных тегов и элементов XML в таком случае в оперативной памяти строится дерево перед просмотром. Эти парсеры проще в реализации, но создание дерева требует большого объёма оперативной памяти. Необходимость частого просмотра узлов дерева сильно замедляет работу такого парсера.

Инициализация парсера происходит похожим образом с SAX-парсером:

**DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();**

**File xml = new File(“data.xml”);**

**Document doc = null;**

**try{**

**DocumentBuilder db = dbf.newDocumentBuilder();**

**doc=db.parse(xml);**

**}catch(Exception e){**

**e.printStackTrace();**

**};**

После чего мы получаем объект класса Document, который представляет собой документ xml в виде дерева (коллекции узлов). Для работы с ним определены классы, представляющие собой различные типы узлов.

Element – базовый блок xml. Имеют потомков, текстовые узлы и их комбинации. В отличии от остальных типов узлов, могут иметь атрибуты, которые хранятся в NamedNodeMap.

Attr – атрибут узла, имеет имя и значение, но не рассматривается как потомок элемента.

Текстовые узлы – просто текстовая информация ил пробельные символы. Данный тип узла имеет служебное имя #text, а его значение – это, собственно, его текстовое содержимое.

Node – базоый тип для всех типов узлов.

Объект класса Document может выдавать много различной информации о содержимом документа – его корневой элемент, элементы по имени тега, по идентификатору, потомки атрибуты, типы узлов, их имена, значения и т.д.

Кроме того, при помощи Dom-парсера можно изменять содержимое документа: создавать новые узлы, заменять их, удалять.

Для переведения дом-дерева в xml-документ необходимо использовать трансформер, которому можно передавать различные настройки формирования документа.

Можно рассмотреть это на создании нового документа при помощи DOM-парсера:

**DocumentBuilderFactory domFactory;**

**DocumentBuilder db;**

**Document doc;**

**Еlement root;**

**Element subroot;**

**Element element;**

**domFactory = DocumentBuilderFactory.*newInstance*();**

**db = domFactory.newDocumentBuilder();**

**doc = db.newDocument();**

**root = doc.createElement("ROOT");**

**doc.appendChild(root);**

**subroot = doc.createElement("subroot");**

**root.appendChild(subroot);**

**element = doc.createElement(“element”);**

**subroot.appendChild(element);**

**Transformer trf = null;**

**DOMSource src = null;**

**try {**

**trf = TransformerFactory.*newInstance*().newTransformer();**

**trf.setOutputProperty(OutputKeys.*ENCODING*, "UTF-8");**

**trf.setOutputProperty(OutputKeys.*STANDALONE*, "yes");**

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

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

**doc.setXmlStandalone(true);**

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

**StreamResult result = new StreamResult(new File("data.xml"));**

**trf.transform(src, result);**

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

**e.printStackTrace();**

**}**

1. *JavaBeans* компоненти. Підтримка простих властивостей та властивостей з обмеженнями (simple, constrained properties). (*10 балів*)
2. Основи роботи з TCP сокетами. Основні етапи створення мережевого додатку за допомогою TCP сокетів. Приклад простого TCPсервера та клієнта: клієнт відправляє рядок на сервер, сервер повертає рядок великими літерами. (*10 балів*)
3. Основи *RMI*. Наведіть приклад простого розподіленого додатку, виконаного за технологією *RMI*, в якому клієнт надає серверу рядок, а сервер повертає клієнту цей рядок подвоєним. (*10 балів*)