Ярослав

40 уровень

Днепр

08.05.2018 42912 26

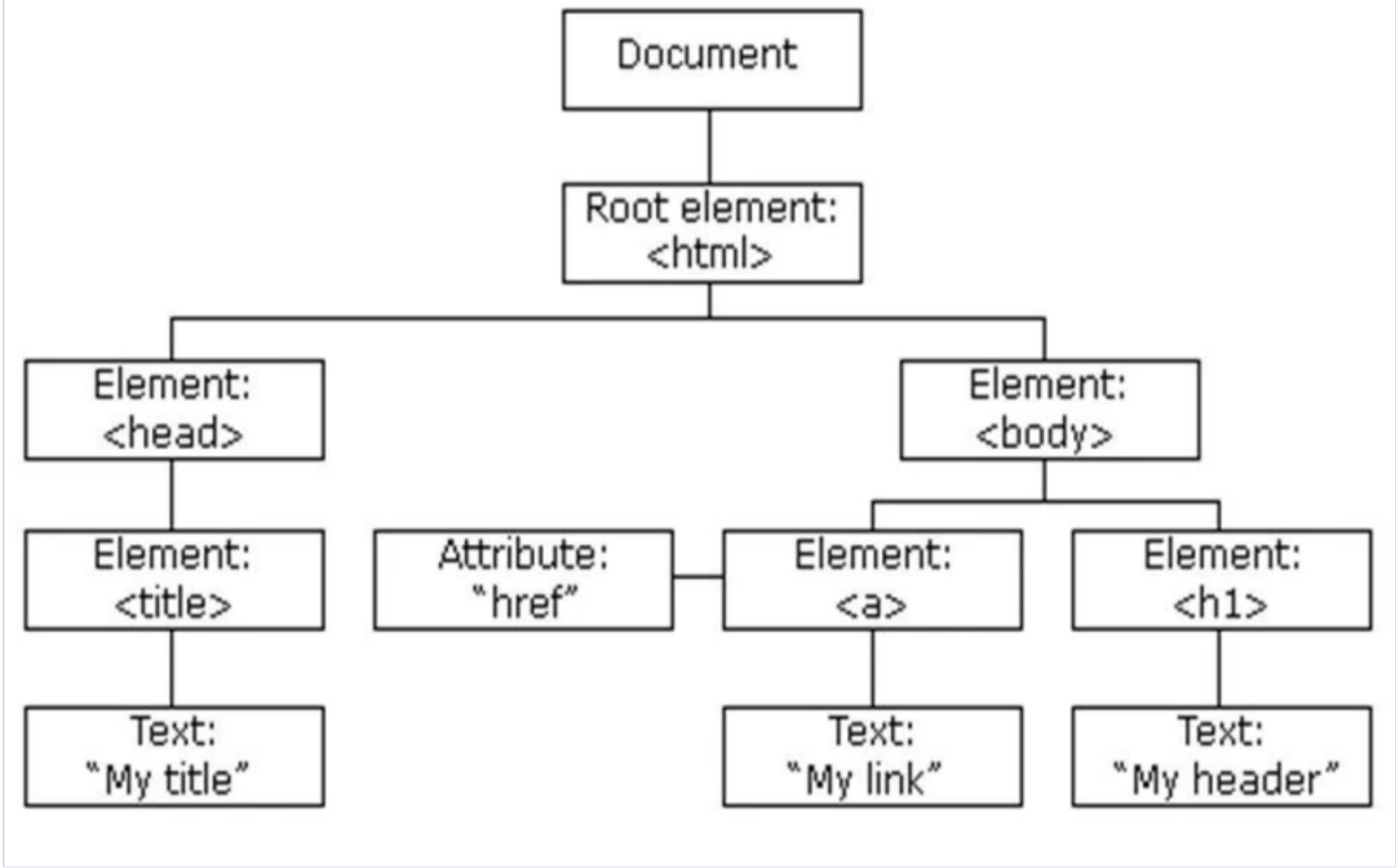
Основы XML для Java программиста. Часть 3.2 из 3 - DOM

Статья из группы Random

1689025 участников

<h2>Вступление</h2>Привет всем читателям статьи, эта часть посвящена DOM. Следующая будет посвящена JAXB и, на этом, цикл основ XML будет завершен. Сначала будет немного теории, а далее только практика. Давайте приступать.

<h2>DOM (Document Object Model) - ТЕОРИЯ</h2>DOM-обработчик устроен так, что он считывает сразу весь XML и сохраняет его, создавая иерархию в виде дерева, по которой мы можем спокойно двигаться и получать доступ к нужным нам элементам.



Таким образом, мы можем, имея ссылку на верхний элемент, получить все ссылки на его внутренние элементы. При чем элементы, которые внутри элемента – дети этого элемента, а он – их родитель.

Однажды считав весь XML в память, мы просто будем путешествовать по его структуре и выполнять нужные нам действия.

Немного уже о программной части DOM в Java: в DOM есть множество интерфейсов, которые созданы, чтобы описывать разные данные. Все эти интерфейсы наследуют один общий интерфейс – Node (узел). Потому, по сути, самый частый тип данных в DOM – это Node (узел), который может быть всем.

У каждого Node есть следующие полезные методы 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` – полезно при обработка пространств имён, чтобы получить имя без префикса.
10. `getTextContent` – возвращает весь текст внутри элемента и всех элементов внутри данного элемента, включая переносы строчек и пробелы.

Примечание по 9 методу: он будет всегда возвращать null, если в DocumentFactory вы не воспользовались методом setNamespaceAware(true), чтобы запустить обработку пространств имён.

Теперь, важная деталь: методы общие для всех Node, но в Node у нас может быть как элемент, так и атрибут. И тут вопросы: какое значение может быть у элемента? Какие производные узлы могут быть у атрибута? И другие не состыковки.

Однако, чтобы самому не пробовать все, в официальных доках есть очень полезная табличка по работе каждого метода в зависимости от типа Node:

Interface	nodeName	nodeValue	attributes
Attr	same as Attr.name	same as Attr.value	null
CDATASection	"#cdata-section"	same as CharacterData.data, the content of the CDATA Section	null
Comment	"#comment"	same as CharacterData.data, the content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	same as DocumentType.name	null	null
Element	same as Element.tagName	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	same as ProcessingInstruction.target	same as ProcessingInstruction.data	null
Text	"#text"	same as CharacterData.data, the content of the text node	null

Качество плохое получилось, так что ссылка на документацию (таблица вверху страницы): Документация Node

Самое важное, что нужно запомнить:

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.

<h2>DOM (Document Object Model) – ПРАКТИКА</h2>В практической части мы будем разбирать разного рода задачки по поиску информации в XML. Так же взяты две задачи из прошлой статьи для сравнения удобства. Давайте начинать, а начать хорошо было бы с импортов:

```
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

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

Даю импорты, чтобы вы не спутали классы :)

Задача №1 – нам нужно достать информацию о всех сотрудниках и вывести её в консоль из следующего XML файла:

```
<?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>
```

```
                <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>
```

Как мы можем видеть, у нас вся информация сохранена в элементах employee. Для того, чтобы где-то хранить её у нас в программе, давайте создадим класс `Employee`:

```
public class Employee {
    private String name, job;

    public Employee(String name, String job) {
        this.name = name;
        this.job = job;
    }

    public String getName() {
        return name;
    }

    public String getJob() {
        return job;
    }
}
```

Теперь, когда у нас есть описание структуры для хранения данных, нам нужна коллекция, которая будет хранить сотрудников. Её мы будем создавать в самом коде. Так же нам нужно создать Document на основе нашего XML:

```
public class DOMExample {
    // Список для сотрудников из XML файла
    private static ArrayList<Employee> employees = new ArrayList<>();

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        // Получение фабрики, чтобы после получить билдер документов.
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

        // Получили из фабрики билдер, который парсит XML, создает структуру Document в виде иерархич
        DocumentBuilder builder = factory.newDocumentBuilder();

        // Запарсили XML, создав структуру Document. Теперь у нас есть доступ ко всем элементам, каки
        Document document = builder.parse(new File("resource/xml_file1.xml"));
    }
}
```

После получения документа, мы обладаем неограниченной властью над всей структурой XML файла. Мы можем получать любые элементы в любое время, возвращаться обратно, чтобы проверить какие-либо данные и, в целом, более гибкий подход, чем был у нас в SAX.

В контексте данной задачи, нам нужно просто извлечь все элементы employee, а после извлечь всю информацию про них.

```java
public class DOMExample {
    // Список для сотрудников из XML файла
    private static ArrayList<Employee> employees = new ArrayList<>();

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        // Получение фабрики, чтобы после получить билдер документов.
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

        // Получили из фабрики билдер, который парсит XML, создает структуру Document в виде иерархич
        DocumentBuilder builder = factory.newDocumentBuilder();

        // Запарсили XML, создав структуру Document. Теперь у нас есть доступ ко всем элементам, каки
        Document document = builder.parse(new File("resource/xml_file1.xml"));

        // Получение списка всех элементов employee внутри корневого элемента (getDocumentElement воз
        NodeList employeeElements = document.getDocumentElement().getElementsByTagName("employee");

        // Перебор всех элементов employee
        for (int i = 0; i < employeeElements.getLength(); i++) {
            Node employee = employeeElements.item(i);

            // Получение атрибутов каждого элемента
            NamedNodeMap attributes = employee.getAttributes();

            // Добавление сотрудника. Атрибут - тоже Node, потому нам нужно получить значение атрибут
            employees.add(new Employee(attributes.getNamedItem("name").getNodeValue(), attributes.get
        }

        // Вывод информации о каждом сотруднике
        for (Employee employee : employees)
            System.out.println(String.format("Информации о сотруднике: имя - %s, должность - %s.", em
    }
}
```

Описание данного решения прямо в решении. Желательно после просмотра кода, вернуться обратно к теории и прочитать еще раз. На самом деле, все понятно инстинктивно. Прочитайте внимательно комментарии и вопросов быть не должно, а если остались – можете написать в комментариях, я отвечу, или в лычку, или просто запустить свою ИДЕЮ и самому попробовать поиграться с кодом, если вы еще этого не сделали.

Таким образом, после запуска кода мы получили следующие выходные данные:

```
Информации о сотруднике: имя - Maksim, должность - Middle Software Developer.
Информации о сотруднике: имя - Ivan, должность - Junior Software Developer.
Информации о сотруднике: имя - Franklin, должность - Junior Software Developer.
Информации о сотруднике: имя - Herald, должность - Middle Software Developer.
Информации о сотруднике: имя - Adam, должность - Middle Software Developer.
Информации о сотруднике: имя - Leroy, должность - Junior Software Developer.
```

Как можно заметить, задача успешно выполнена! Давайте переходить к следующей задаче :)

Задача №2 – вводится с консоли имя элемента, про который нужно вывести информацию об всех элементах внутри его и их атрибутах со следующего XML файла:

```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>
```

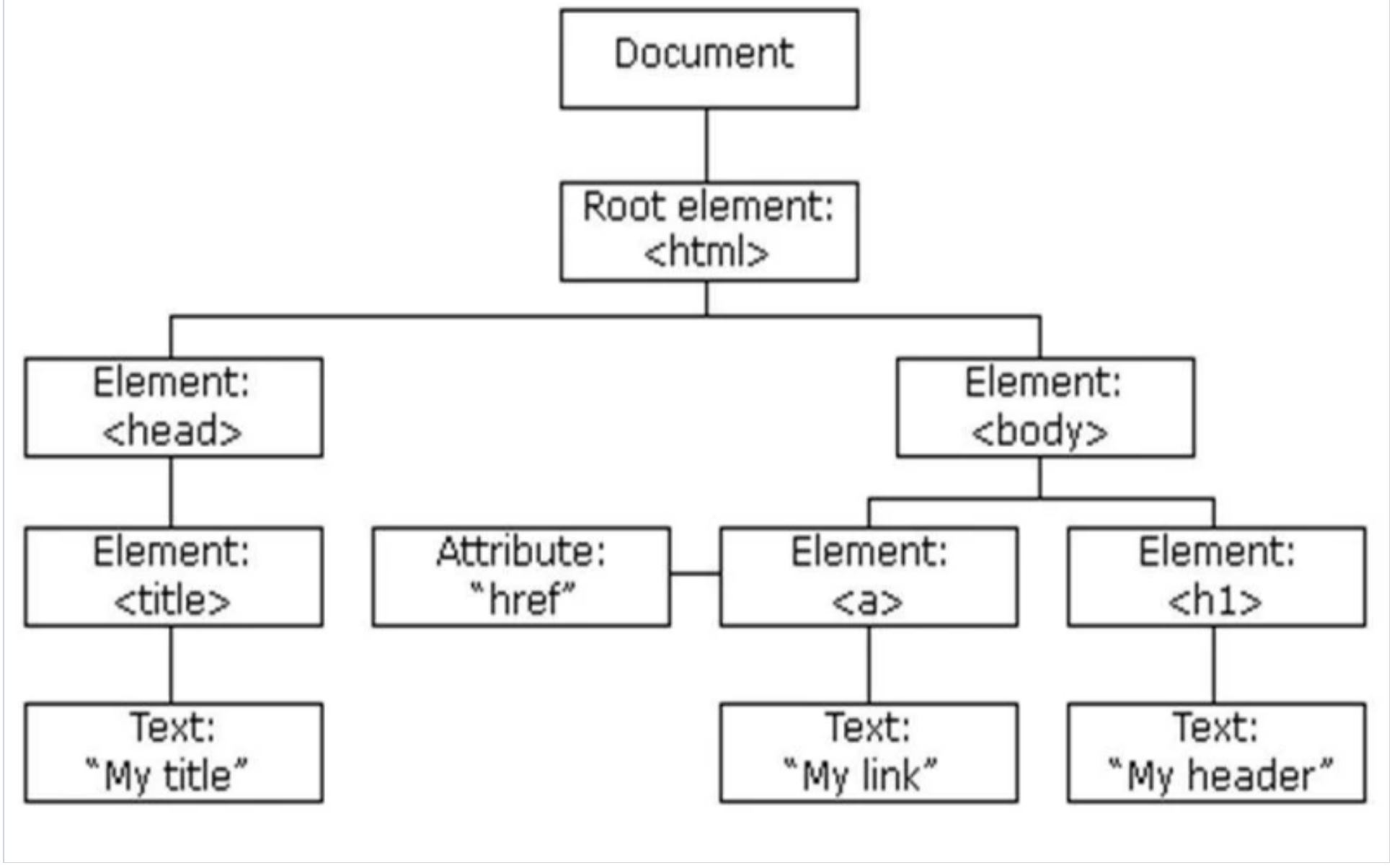
Все достаточно просто: мы должны получить элемент по его имени, которое считаем, а после пройти по всем дочерним узлам. Для этого нужно пройтись по всем дочерним узлам всех дочерних узлов, которые являются элементами.

Решение данной задачи:

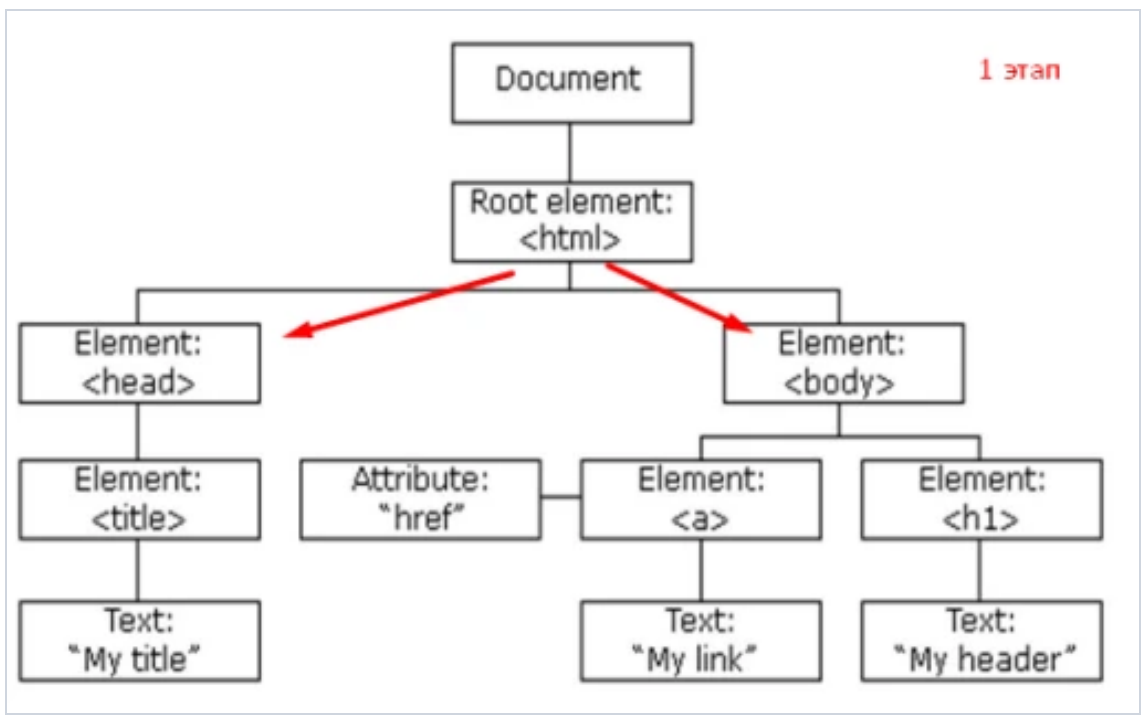
```java
public class DOMExample {
    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        // Ридер для считывания имени тега из консоли
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        // Получение фабрики, чтобы после получить билдер документов.
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

        // Получили из фабрики билдер, который парсит XML, создает структуру Document в виде иерархич
        DocumentBuilder builder = factory.newDocumentBuilder();

        // Запарсили XML, создав структуру Document. Теперь у нас есть доступ ко всем элементам, каки
        Document document = builder.parse(new File("resource/xml_file3.xml"));

        // Считывание имени тега для поиска его в файле
        String element = reader.readLine();

        // Получение списка элементов, однако для удобства будем рассматривать только первое совпаден
        // Так же заметьте, что мы ищем элемент внутри документа, а не рут элемента. Это сделано для
        NodeList matchedElementsList = document.getElementsByTagName(element);

        // Даже если элемента нет, всегда будет возвращаться список, просто он будет пустым.
        // Потому, чтобы утверждать, что элемента нет в файле, достаточно проверить размер списка.
        if (matchedElementsList.getLength() == 0) {
            System.out.println("Тег " + element + " не был найден в файле.");
        } else {
            // Получение первого элемента.
            Node foundedElement = matchedElementsList.item(0);

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

            // Если есть данные внутри, вызов метода для вывода всей информации
            if (foundedElement.hasChildNodes())
```

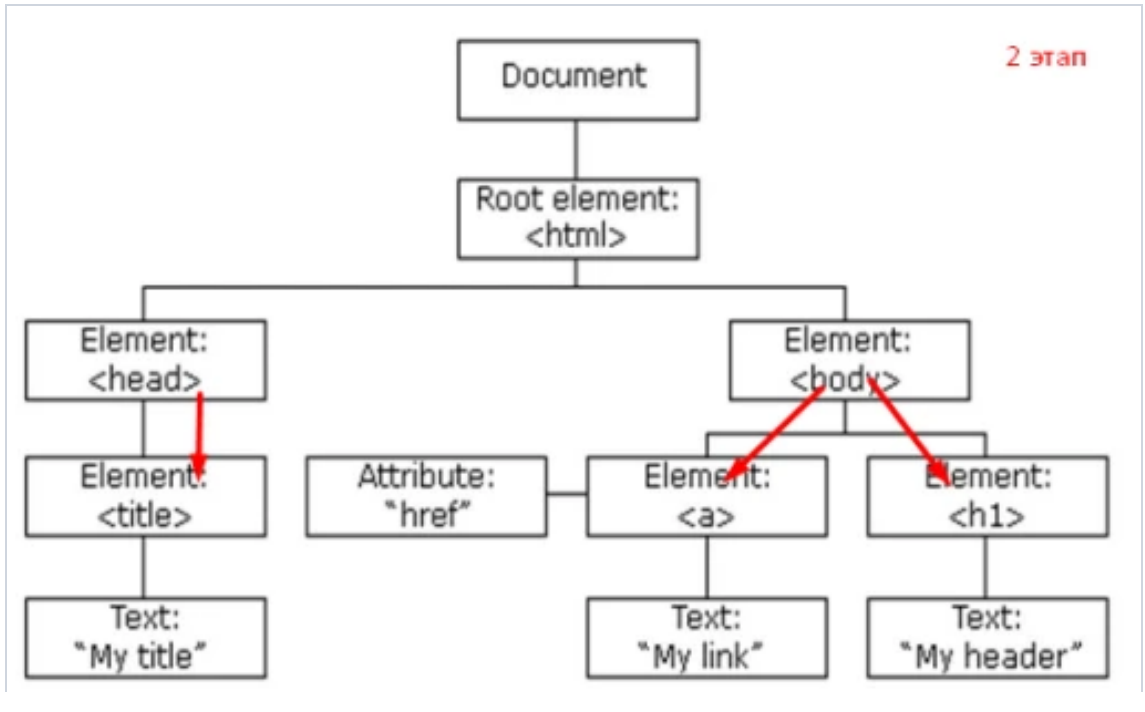
```
    }

    /**
     * Рекурсивный метод, который будет выводить информацию про все узлы внутри всех узлов, которые п
     * @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 нуж
            if (node.getNodeType() == Node.TEXT_NODE) {
                // Фильтрация информации, так как пробелы и переносы строчек нам не нужны. Это не инф
                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 обрабатываем его как элемент.
            else {
                System.out.println("Найден элемент: " + node.getNodeName() + ", его атрибуты:");

                // Получение атрибутов
                NamedNodeMap attributes = node.getAttributes();

                // Вывод информации про все атрибуты
                for (int k = 0; k < attributes.getLength(); k++)
                    System.out.println("Имя атрибута: " + attributes.item(k).getNodeName() + ", его з
            }

            // Если у данного элемента еще остались узлы, то вывести всю информацию про все его узлы.
            if (node.hasChildNodes())
                printInfoAboutAllChildNodes(node.getChildNodes());
        }
    }
}
```

Все описание решения есть в комментариях, однако хотелось бы немного изобразить графически подход, который мы задействовали, на примере картинки из теории.

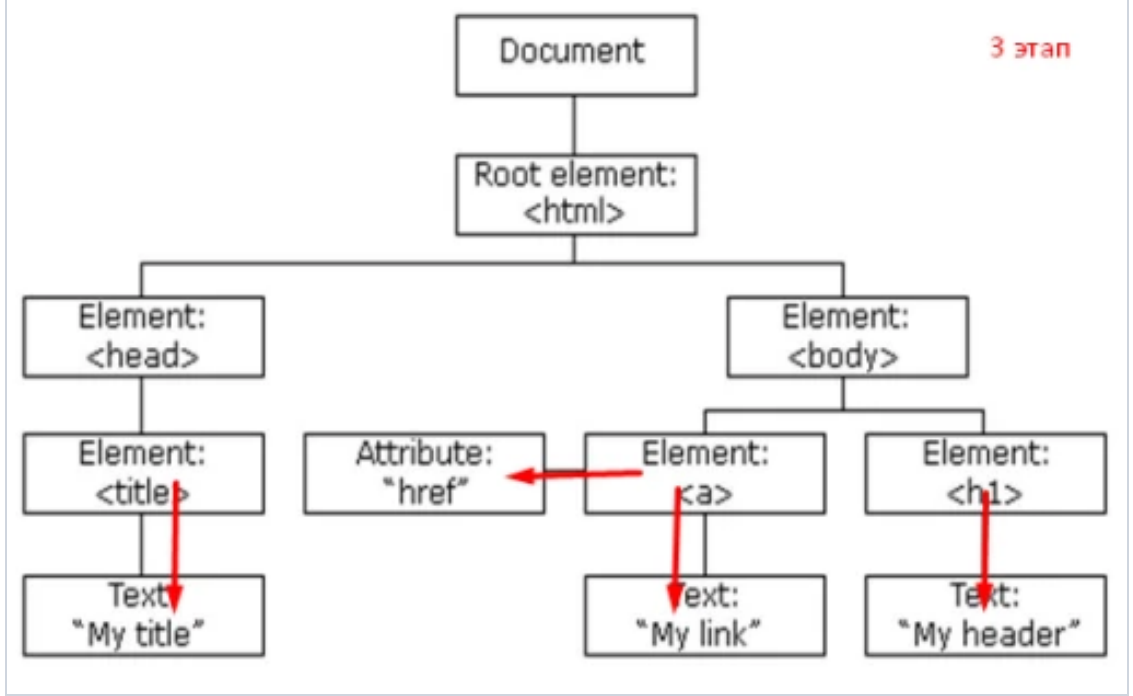


Будем считать, что информацию нам нужно вывести про тег html. Как вы видите, нам нужно идти сверху-вниз от корня дерева. Все линии – это узлы.





НАЧАТЬ ОБУЧЕНИЕ



В решении мы рекурсивно будем идти от начала нужного элемента по всем его узлам, и если какой-то из его узлов – элемент, то мы перебираем так же все узлы этого элемента.

Таким образом, после запуска кода мы получили следующие выходные данные для элемента root:

```
Элемент был найден!
Найден элемент: oracle, его атрибуты:
Найден элемент: connection, его атрибуты:
Имя атрибута: value, его значение: jdbc:oracle:thin:@10.220.140.48:1521:test1
Найден элемент: user, его атрибуты:
Имя атрибута: value, его значение: secretOracleUsername
Найден элемент: password, его атрибуты:
Имя атрибута: value, его значение: 111
Найден элемент: mysql, его атрибуты:
Найден элемент: connection, его атрибуты:
Имя атрибута: value, его значение: jdbc:mysql:thin:@10.220.140.48:1521:test1
Найден элемент: user, его атрибуты:
Имя атрибута: value, его значение: secretMySQLUsername
Найден элемент: password, его атрибуты:
Имя атрибута: value, его значение: 222
```

Задача успешно решена!

Задача №3 – из следующего XML файла, где сохранена информация про студентов, профессоров и сотрудников, нужно считать информацию и вывести её в консоль:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<database>
    <students>
        <student name="Maksim" course="3" specialization="CE" />
        <student name="Stephan" course="1" specialization="CS" />
        <student name="Irvin" course="2" specialization="CE" />
    </students>

    <professors>
        <professor name="Herald" experience="7 years in University" discipline="Math" />
        <professor name="Adam" experience="4 years in University" discipline="Programming" />
        <professor name="Anton" experience="6 years in University" discipline="English" />
    </professors>

```

НАЧАТЬ ОБУЧЕНИЕ

```
        <member name="Jordan" position="janitor" />
        <member name="Mike" position="janitor" />
    </service>
</database>
```

Задача довольно простая, однако интересная. Для начала, нам нужно создать 4 класса: сотрудника, профессора и студента, а так же общий абстрактный класс Human, чтобы вынести переменную name из каждого класса под общий знаменатель:

Абстрактный класс-родитель

```
public abstract class Human {
    private String name;

    public Human(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Студент

```
public class Student extends Human {
    private String course, specialization;

    public Student(String name, String course, String specialization) {
        super(name);
        this.course = course;
        this.specialization = specialization;
    }

    public String getCourse() {
        return course;
    }

    public String getSpecialization() {
        return specialization;
    }

    public String toString() {
        return "Голодный студент " + getName() + " " + course + "-го курса, обучающийся по специальнс
    }
}
```

Профессор

```
public class Professor extends Human {
    private String experience, discipline;

    public Professor(String name, String experience, String discipline) {
```

```
        this.discipline = discipline;
    }

    public String getExperience() {
        return experience;
    }

    public String getDiscipline() {
        return discipline;
    }

    public String toString() {
        return "Профессор " + getName() + ", обладающий опытом: \"" + experience + "\", выкладает дис
    }
}
```

Сотрудник

```
public class Member extends Human {
    private String position;

    public Member(String name, String position) {
        super(name);
        this.position = position;
    }

    public String getPosition() {
        return position;
    }

    public String toString() {
        return "Сотрудник обслуживающего персонала " + getName() + ", должность: " + position;
    }
}
```

Теперь, когда наши классы готовы, нам достаточно написать код для получения всех элементов student, professor и member, а потом достать их атрибуты.

Для хранения мы будем использовать коллекцию, которая будет хранить объекты общего для всех родительского класса – Human.

И так, решение данной задачи:

```
public class DOMExample {
    // Коллекция для хранения всех людей
    private static ArrayList<Human> humans = new ArrayList<>();

    // Константы для элементов
    private static final String PROFESSOR = "professor";
    private static final String MEMBER = "member";
    private static final String STUDENT = "student";

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
```

```
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

        // Получили из фабрики билдер, который парсит XML, создает структуру Document в виде иерархич
        DocumentBuilder builder = factory.newDocumentBuilder();

        // Запарсили XML, создав структуру Document. Теперь у нас есть доступ ко всем элементам, каки
        Document document = builder.parse(new File("resource/xml_file3.xml"));

        // Получение информации про каждый элемент отдельно
        collectInformation(document, PROFESSOR);
        collectInformation(document, MEMBER);
        collectInformation(document, STUDENT);

        // Вывод информации
        humans.forEach(System.out::println);
    }

    /**
     * Метод ищет информацию про теги по имени element и вносит всю информацию в коллекцию humans.
     * @param document Документ, в котором будем искать элементы.
     * @param element Имя элемента, теги которого нужно найти. Должна быть одна из констант, которые
     */
    private static void collectInformation(Document document, final String element) {
        // Получение всех элементов по имени тега.
        NodeList elements = document.getElementsByTagName(element);

        // Перебор всех найденных элементов
        for (int i = 0; i < elements.getLength(); i++) {
            // Получение всех атрибутов элемента
            NamedNodeMap attributes = elements.item(i).getAttributes();
            String name = attributes.getNamedItem("name").getNodeValue();

            // В зависимости от типа элемента, нам нужно собрать свою дополнительну информацию про ка
            switch (element) {
                case PROFESSOR: {
                    String experience = attributes.getNamedItem("experience").getNodeValue();
                    String discipline = attributes.getNamedItem("discipline").getNodeValue();

                    humans.add(new Professor(name, experience, discipline));
                } break;
                case STUDENT: {
                    String course = attributes.getNamedItem("course").getNodeValue();
                    String specialization = attributes.getNamedItem("specialization").getNodeValue();

                    humans.add(new Student(name, course, specialization));
                } break;
                case MEMBER: {
                    String position = attributes.getNamedItem("position").getNodeValue();

                    humans.add(new Member(name, position));
                } break;
            }
        }
    }
```

Заметьте, что нам достаточно только названия элемента, чтобы получить вообще все эти элементы из документа. Это значительно упрощает процесс поиска нужной информации.

Вся информация про код помещена в комментарии. Нового ничего не было использовано, чего не было бы в предыдущих задачах.

Выходные данные кода:

```
Профессор Herald, обладающий опытом: "7 years in University", выкладает дисциплину Math
Профессор Adam, обладающий опытом: "4 years in University", выкладает дисциплину Programming
Профессор Anton, обладающий опытом: "6 years in University", выкладает дисциплину English
Сотрудник обслуживающего персонала John, должность: janitor
Сотрудник обслуживающего персонала Jordan, должность: janitor
Сотрудник обслуживающего персонала Mike, должность: janitor
Голодный студент Maksim 3-го курса, обучающийся по специальности CE
Голодный студент Stephan 1-го курса, обучающийся по специальности CS
Голодный студент Irvin 2-го курса, обучающийся по специальности CE
```

Задача решена!

+122

Комментарии (26) популярные новые старые

JavaCoder

Введите текст комментария

Anna Gubkina Уровень 16, San Francisco 13 марта, 20:14

Отличная статья! Стало многое понятно, спасибо❤

Ответить 0

Jerry Backend Developer 11 января, 17:48

JAXB можно уже не ждать?

Ответить +1

LuneFox инженер по сопровождению в **BIFIT** EXPERT 6 января, 02:51

Неплохо было бы заменить название переменной **foundedElement** на **foundElement**, элемент ведь нашли, а не основали :)

Ответить 0

PaiMei in J# Grand Master в **Eagles' Claw** 28 октября 2021, 13:33

Отличный цикл статей, отдельное спасибо за примеры

Ответить 0

Андрей Гузанов Уровень 37, Новосибирск, Россия 9 октября 2021, 10:10

Очень крутой перечень тем, а главное, всё понятно расписано.
Спасибо !

Ответить 0

barracuda Уровень 41, Санкт-Петербург, Россия EXPERT 2 февраля 2021, 23:20

НАЧАТЬ ОБУЧЕНИЕ

Ответить 0

itsmymainaim Уровень 18 — 23 октября 2020, 00:03

как быть если у меня есть ссылка на этот xml файл,допустим на курсы валют цб,как через ссылку это все сделать?

Ответить 0

Ярослав Java Developer MASTER — 29 октября 2020, 00:39

Привет, мало инфы, какая ссылка? Ссылка - это лишь адрес, но ты не уточнил протокол (http, ftp) и формат (обычный ли это .xml файл или сгенеренная html страница, где есть текст в формате xml), без этого неясно.
Но в целом, в случае с http и что это обычная хмлка чистая, то спокойно через какой-то http клиент делаешь запрос, тебе приходит хмлка допустим в какой-то String, и потом через любой инструмент манипулируешь этими данными (sax, dom, jaxb)

если хочешь нормальный http клиент подключить отдельно, то HttpOk довольно известный, можно его. Или есть в самой жаве пакет java.net

java net
send request

Ответить 0

максим Уровень 40, Екатеринбург — 24 августа 2020, 16:30

Печаль, так и не увидели мы JAXB

Ответить +4

VDT Java Developer в **USETECH** — 18 июня 2020, 17:46

Читать и обрабатывать класс... Сюда бы еще пример сохранения результата (для полной картины работы с XML). А в случае с SAX, дак там еще и форматирование соблюсти)

Ответить 0

hidden #1281202 Уровень 41 — 28 июля 2019, 18:34

Очень хорошо! Вторую лекцию постичь было труднее всего, но это намного лучше бесконечных томов воды.

Ответить +2

Показать еще комментарии

Русский