Ярослав

40 уровень

Днепр

07.05.2018 41794 17

Основы XML для Java программиста – Часть 3.1 из 3 - SAX

Статья из группы Random

1689031 участник

Вступление

Привет всем читателям моей еще не последней статьи и хочу поздравить: сложное про XML осталось позади. В данной статье будет уже код на Java. Будет немного теории, а далее практика.

Из-за того, что одного материала по SAX у меня получилось на 10 страничек в ворде, я понял, что в лимиты не помещусь. Потому, 3 статья будет разделена на 3 отдельные статьи, как бы это странно не звучало. Будет все в таком порядке: SAX -> DOM -> JAXB.

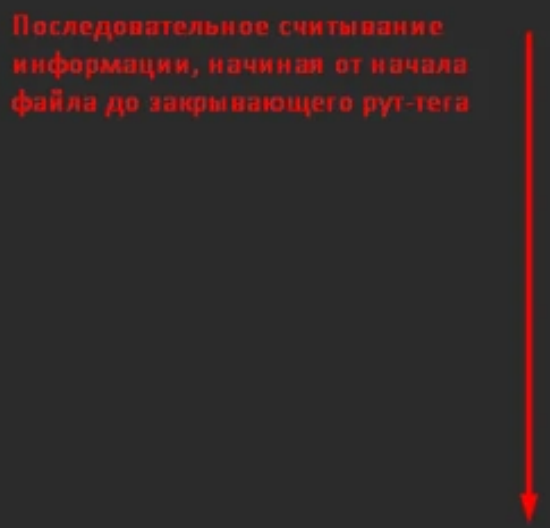
Данная статья будет посвящена только SAX.

P.S. Там где-то в курсе была задача, где надо было в HTML файле вывести все внутренние элементы. После данной статьи, вы сможете это сделать без считывания построчно обычным `BufferedReader` и сложными алгоритмами обработки, а, так же, близкое решение будет дано в последнем практическом примере. Давайте приступать :)

SAX (Simple API for XML) — ТЕОРИЯ

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

```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>
```



У него есть немало событий, однако самые частые и полезные следующие:

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**.

Преимущества: высокая производительность благодаря "прямому" способу считывания данных, низкие затраты памяти.
Недостатки: ограниченная функциональность, а, значит, в нелинейных задачах дорабатывать её надо будет уже нам.

SAX (Simple API for XML) – ПРАКТИКА

Сразу список импортов, чтобы вы не искали и ничего не спутали:

```
1  import org.xml.sax.Attributes;
2  import org.xml.sax.SAXException;
3  import org.xml.sax.helpers.DefaultHandler;
4
5  import javax.xml.parsers.ParserConfigurationException;
6  import javax.xml.parsers.SAXParser;
7  import javax.xml.parsers.SAXParserFactory;
```

Теперь, для начала, нам нужно создать SAXParser:

```
1  public class SAXExample {
2      public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
3          // Создание фабрики и образца парсера
4          SAXParserFactory factory = SAXParserFactory.newInstance();
5          SAXParser parser = factory.newSAXParser();
6      }
7  }
```

Как вы видите, сначала нужно создать фабрику, а потом в фабрике создать уже сам парсер. Теперь, когда у нас есть сам парсер, нам нужен обработчик его событий. Для этого нам нужен отдельный класс ради нашего же удобства:

```
1  public class SAXExample {
2      public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
```

```
    }

    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) t
            // 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 ст
        }
    }
}
```

Мы создали класс со всеми нужными нам методами для обработки событий, которые были перечислены в теории.

Еще немного дополнительной теории:

Немного про `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.

О методах `startElement` и `endElement`: `uri` — это пространство, в котором находится элемент, `localName` — это имя элемента без префикса, `qName` — это имя элемента с префиксом (если он есть, иначе просто имя элемента).

`uri` и `localName` всегда пустые, если мы не подключили в фабрике обработку пространств. Это делается методом фабрики `setNamespaceAware(true)`. Тогда мы сможем получать пространство (`uri`) и элементы с префиксами перед ними (`localName`).

Задача №1 — у нас есть следующий XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<company>
    <name>IT-Heaven</name>
```

```
            <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>
```

Наша цель: достать всю информацию про всех сотрудников из данного файла.

Для начала, нам нужно создать 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;
    }
}
```

А в нашем основном классе `SAXExample` нам нужен список со всеми сотрудниками:

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

Теперь давайте внимательно смотреть, где нужная нам информация находится в XML файле. И, как мы можем видеть, вся нужная нам информация — это атрибуты элементов `employee`. А так, как `startElement` у нас обладает таким полезным параметром, как `attributes`, то у нас довольно простая задача.

Для начала, давайте уберем ненужные методы, чтобы не захламлять наш код. Нам нужен только метод `startElement`. А в самом методе мы должны собрать информацию с атрибутов тега employee. Внимание:

```
public class SAXExample {
    private static ArrayList<Employee> employees = new ArrayList<>();

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

```
        SAXParser parser = factory.newSAXParser();
    }

    private static class XMLHandler extends DefaultHandler {
        @Override
        public void startElement(String uri, String localName, String qName, Attributes attributes) t
            if (qName.equals("employee")) {
                String name = attributes.getValue("name");
                String job = attributes.getValue("job");
                employees.add(new Employee(name, job));
            }
        }
    }
}
```

Логика простая: если имя элемента — `employee`, мы просто будем получать информацию про его атрибуты. В `attributes` есть полезный метод, где, зная название атрибута, можно получить его значение. Именно его мы и использовали.

Теперь, когда мы создали обрабатывание события на начало элемента, **нам нужно запарсить наш XML файл**. Для этого достаточно сделать так:

```
public class SAXExample {
    private static ArrayList<Employee> employees = new ArrayList<>();

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        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.getNam
    }

    private static class XMLHandler extends DefaultHandler {
        @Override
        public void startElement(String uri, String localName, String qName, Attributes attributes) t
            if (qName.equals("employee")) {
                String name = attributes.getValue("name");
                String job = attributes.getValue("job");
                employees.add(new Employee(name, job));
            }
        }
    }
}
```

В методе parse вы должны передать путь к xml файлу и обработчик, который вы создали. И так, с помощью данного кода мы достали информацию из этого XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<company>
    <name>IT Heaven</name>
```

```
        <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>
```

А выходные данные мы получили такие:

```
Имя сотрудника: Maksim, его должность: Middle Software Developer
Имя сотрудника: Ivan, его должность: Junior Software Developer
Имя сотрудника: Franklin, его должность: Junior Software Developer
Имя сотрудника: Herald, его должность: Middle Software Developer
Имя сотрудника: Adam, его должность: Middle Software Developer
Имя сотрудника: Leroy, его должность: Junior Software Developer
```

Задача выполнена!

Задача №2 — у нас есть следующий 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>
```

```
                    <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>
```

Наша цель: достать всю информацию про всех сотрудников из данного файла.

Задача хорошо продемонстрирует, каким образом плохо структурированный XML файл может приводить к усложнению написания кода.

Как вы видите, информация про имя и должность теперь хранится как текстовая информация внутри элементов `name` и `job`. Для считывания текста внутри элементов у нас есть метод characters.

Для этого, нам нужно создать новый класс-обработчик с улучшенной логикой. Не забывайте, что обработчики – полноценные классы, способные хранить в себе логику любой сложности. Потому, сейчас мы будем тюнинговать наш обработчик.

На самом деле, достаточно заметить, что у нас всегда `name` и `job` идут по очереди, и не важно, в каком порядке, мы можем спокойно сохранить имя и профессию в отдельные переменные, и когда обе переменные сохранены – создать нашего сотрудника. Только вот вместе с началом элемента у нас нет параметра для текста внутри элемента. Нам нужно использовать методы для текста.

Но как нам получить текстовую информацию внутри элемента, если это совершенно разные методы? Мое решение: нам достаточно запомнить имя последнего элемента, а в `characters` проверять, в каком элементе мы считываем информацию. **Так же нужно помнить, что <codee>characters считывает все символы внутри элементов, а это значит, что будут считываться все пробелы и даже переносы строчек. А они нам не нужны. Нам нужно игнорировать эти данные, так как они неправильные.</codee>** Код:

```
public class SAXExample {
    private static ArrayList<Employee> employees = new ArrayList<>();

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = factory.newSAXParser();

        AdvancedXMLHandler handler = new AdvancedXMLHandler();
        parser.parse(new File("resource/xml_file2.xml"), handler);

        for (Employee employee : employees)
            System.out.println(String.format("Имя сотрудника: %s, его должность: %s", employee.getNam
    }

    private static class AdvancedXMLHandler extends DefaultHandler {
        private String name, job, lastElementName;
```

```
        public void startElement(String uri, String localName, String qName, Attributes attributes) t
            lastElementName = qName;
        }

        @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;
            }
        }

        @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;
            }
        }
    }
}
```

Как вы видите, из-за банального усложнения структуры XML файла у нас значительно усложнился код. Однако, код не сложный.

Описание: мы создали переменные для хранения данных про сотрудника (`name`, `job`), а так же переменную `lastElementName`, чтобы фиксировать, внутри какого элемента мы находимся. После этого, в методе `characters` мы фильтруем информацию, и 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.

Выходные данные решения эквивалентны первому примеру:

```
Имя сотрудника: Maksim, его должность: Middle Software Developer
Имя сотрудника: Ivan, его должность: Junior Software Developer
Имя сотрудника: Franklin, его должность: Junior Software Developer
Имя сотрудника: Herald, его должность: Middle Software Developer
Имя сотрудника: Adam, его должность: Middle Software Developer
Имя сотрудника: Leroy, его должность: Junior Software Developer
```

Таким образом, данная задача была **решена**, но вы можете заметить то, что сложность выше. Потому можно сделать вывод, что хранить текстовую информацию в атрибутах чаще всего будет правильней, чем в отдельных элементах.

И еще одна сладкая задача, которая будет частично решать задачу на JavaRush про вывод информации об элементе в HTML, только её надо будет немного подредактировать, тут мы будем просто перечислять все элементы внутри какого-то элемента :)

Для данной задачи мы будем использовать следующий XML файл:

```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>
```

Как вы видите, у нас тут есть три возможных сценария: `root`, `mysql`, `oracle`. Тогда программа будет выводить всю инфу о всех элементах внутри. Как же нам сделать такое? А достаточно просто: нам достаточно объявить логическую переменную `isEntered`, которая будет означать, внутри ли мы нужно нам элемента, и если внутри – считывать все данные из `startElement`. **Код решения:**

```java
public class SAXExample {
    private static boolean isFound;

    public static void main(String[] args) throws ParserConfigurationException, SAXException, IOExcep
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = factory.newSAXParser();

        SearchingXMLHandler handler = new SearchingXMLHandler("root");
        parser.parse(new File("resource/xml_file3.xml"), handler);

        if (!isFound)
            System.out.println("Элемент не был найден.");
    }

    private static class SearchingXMLHandler extends DefaultHandler {
        private String element;
        private boolean isEntered;

        public SearchingXMLHandler(String element) {
            this.element = element;
        }

        @Override
        public void startElement(String uri, String localName, String qName, Attributes attributes) t
            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
```

```
            if (qName.equals(element)) {
                isEntered = true;
                isFound = true;
            }
        }

        @Override
        public void endElement(String uri, String localName, String qName) throws SAXException {
            if (qName.equals(element))
                isEntered = false;
        }
    }
}
```

В данном коде мы при входе в элемент, про который нам нужна информация, выставляем флажок `isEntered` в true, что значит, что мы внутри элемента. И как только мы оказались внутри элемента, мы просто каждый новый элемент в `startElement` обрабатываем, зная, что он точно внутренний элемент нашего элемента. Таким образом, мы выводим имя элемента и его название. Если же элемент не был найден в файле, то у нас есть переменная `isFound`, которая устанавливается тогда, когда элемент находится, и если она false, то будет выведено сообщение, что элемент не найден.

И как вы видите, в примере в конструктор `SearchingXMLHandler` мы передали `root` элемент. Вывод для него:

```
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
```

Таким образом, мы получили всю информацию про внутренние элементы и их атрибуты. **Задача решена.** <h2>Эпилог</h2>Вы ознакомились, что SAX довольно интересный инструмент и вполне эффективный, и его можно использовать по-разному, с разными целями и так далее, достаточно только посмотреть на задачу с правильной стороны, как это показано в задаче №2 и №3, где SAX не предоставлял прямых методов для решения задачи, но, благодаря нашей смекалке, у нас получилось придумать выход из ситуации.

Следующая часть статьи будет целиком посвящена DOM. Надеюсь, что вам было интересно познакомиться с SAX. Поэкспериментируйте, попрактикуйтесь и вы поймете, что все довольно просто. А на этом все, удачи вам в программировании и ждите скоро часть про DOM. **Успехов вам в обучении :)**

Предыдущая статья: [Конкурс] Основы XML для Java программиста - Часть 2 из 3
Следующая статья: [Конкурс] Основы XML для Java программиста - Часть 3.2 из 3 - DOM

+112

Комментарии (17)

популярные новые старые

JavaCoder

Введите текст комментария

Александр Горохов Уровень 25, Дятьково, Россия — 22 июля, 18:30

Большое спасибо за статьи и приведённые примеры

Ответить 0

Joker Уровень 11, Вильнюс, Belarus — 13 мая, 17:52

Делал первый пример, в консоль 6 раз выводит последнего работника. Думал, что уже где-то накосячил и простоя взял и скопировал код. В итоге тоже самое... Я один такой или в коде какой-то баг???

Ответить 0

Joker Уровень 11, Вильнюс, Belarus — 14 мая, 14:32

Во втором примере то же самое, просто выводит в консоль последнего работника 6 раз... Может кто-то подсказать, это у меня проблема или в примере ошибка?

Ответить 0

Александр Горохов Уровень 25, Дятьково, Россия — 22 июля, 13:41

Всё нормально выводит, только что проверял

Ответить 0

Essah King Уровень 37, Чернигов, Украина — 4 февраля, 18:08

Попробовал на другом примере
В конце каждой строчки выводит я так понял хеш код
Название: Бельгийские Вафли.java.io.PrintStream@610455d6
Цена: $5.95.java.io.PrintStream@610455d6
Описание: две известных Бельгийских Вафли с обилием настоящего кленового сиропаjava.io.PrintStream@610455d6
Калорийность: 650java.io.PrintStream@610455d6
подскажите как можно от этого избавиться?))

Ответить 0

LuneFox инженер по сопровождению в **BIFIT** EXPERT — 8 января, 20:55

с м е к а л о ч к а

Ответить 0

Vladislav Klimenko Backend Developer в **Эвотор** — 1 марта 2021, 13:14

Хорошо пишешь, статья наглядная, на ее основе решил пару задач на работе. Спасибо!

Ответить 0

Алексей Уровень 37, Санкт-Петербург — 19 февраля 2021, 13:00

Спасибо

Ответить 0

Лейтенант Ден Уровень 31, Москва, Россия — 20 октября 2020, 12:33

Материал этой статьи это Эверест. Судя по количеству лайков и комментов - доходят сюда самые стойкие

Ответить 0

Хорс Уровень 41, Харьков — 27 июля 2020, 12:33

интересно конечно. Но сам я такое не наваяю

Ответить 0

Dmitry Уровень 0 — 11 января 2020, 05:27

Алексей Уровень 26, Москва 10 декабря 2019, 22:55

Супер серия статей

Ответить +2

Показать еще комментарии