нетология

Работа с файлами CSV, XML, JSON

Григорий
Вахмистров

Григорий Вахмистров

Backend Developer в Tennisi.bet

План занятия

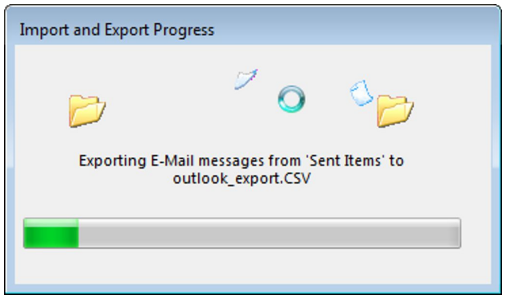
1. Работа с CSV
2. Работа с XML
3. Работа с JSON
4. Итоги
5. Домашнее задание

Работа с CSV

Что такое CSV файлы

CSV означает “*comma-separated values*” - значения, разделенные запятыми.

Формат **CSV** используется для создания файлов для экспорта или импорта данных.



Таким образом могут передаваться контакты, емейлы, таблицы, например:
Исходный текст:

```
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture «Extended Edition»","",4900.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
```

Результирующая таблица:

1997	Ford	E350	ac, abs, moon	3000
1999	Chevy	Venture «Extended Edition»		4900
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799

Библиотека OpenCSV

Язык Java не предоставляет собственного эффективного способа работы с файлами CSV. Поэтому рекомендуется использовать сторонние инструменты и библиотеки.

OpenCSV — библиотека парсеров CSV файлов, поддерживающая весь основной функционал для работы с CSV.

Наиболее полный ***users guide*** можно найти на сайте разработчиков:
http://opencsv.sourceforge.net/

Подключаем OpenCSV

В зависимости от того используем ли мы сборщик проекта и какой именно сборщик, есть несколько способов подключить библиотеку:

- Если используем **Maven**, то подключаем зависимость в **pom.xml** файле:

```
<dependency>
  <groupId>net.sf.opencsv</groupId>
   <artifactId>opencsv</artifactId>
   <version>2.3</version>
</dependency>
```

- Если используем **Gradle**, то подключаем зависимость в **gradle.build** файле:

```
compile 'com.opencsv:opencsv:5.1'
```

- Если не используем сборщик, то загружаем jar-файлы из http://sourceforge.net/projects/opencsv/ и подключаем его вручную

Некоторые полезные классы

Ниже приведен список наиболее используемых классов OpenCSV:

- CSVParser: простой CSV-парсер. Реализует разделение одной строки на поля.
- CSVReader: предназначен для чтения CSV-файла из Java-кода.
- CSVWriter: предназначен для записи CSV-файла из Java-кода.
- CsvToBean: создает Java объекты из содержимого CSV-файла.
- BeanToCsv: экспортирует данные Java объекты в CSV-файл.
- ColumnPositionMappingStrategy: используется CsvToBean (или BeanToCsv) для импорта CSV-данных, если требуется сопоставление полей CSV с полями Java класса.

Создание нового CSV-файла

Создадим экземпляр класса CSVWriter и запишем данные в файл CSV:

```
// Создаем запись
String[] employee = "1,David,Miller,Australia,30".split(",");
// Создаем экземпляр CSVWriter
try (CSVWriter writer = new CSVWriter(new FileWriter("staff.csv"))) {
   // Записываем запись в файл
   writer.writeNext(employee);
} catch (IOException e) {
   e.printStackTrace();
}
```

В корне проекта появится файл **staff.csv** с следующим содержимым:

```
"1","David","Miller","Australia","30"
```

Чтение и парсинг CSV-файла

Как уже упоминалось выше, для чтения CSV-файла необходимо использовать **CSVReader**. Давайте посмотрим на простой пример для чтения файла CSV:

```
// Создаем экземпляр CSVWriter
// Разделитель по умолчанию - запятая
// Символ выражения по умолчанию - двойные кавычки
try (CSVReader reader = new CSVReader(new FileReader("staff.csv"))) {
   // Массив считанных строк
   String[] nextLine;
   // Читаем CSV построчно
   while ((nextLine = reader.readNext()) != null) {
       // Работаем с прочитанными данными.
       System.out.println(Arrays.toString(nextLine));
   }
} catch (IOException | CsvValidationException e) {
   e.printStackTrace();
}
```

Вышеприведенный пример читает файл CSV по одной строке и печатает в консоль:.

```
[1, David, Miller, Australia, 30]
```

Чтение и парсинг CSV-файла

Можно прочитать весь CSV-файл за один раз, а затем перебрать данные по своему усмотрению. Ниже приведен пример чтения CSV-данных с использованием метода **readAll()**.

```
// Читаем все строки за один раз
List<String[]> allRows = reader.readAll();
// Пройдемся по массиву строк
for (String[] row : allRows) {
   // Выполним операцию над записью
   System.out.println(Arrays.toString(row));
}
```

> В приведенном выше примере считывается весь CSV-файл, а затем строки перебираются по очереди.

Расширение существующего CSV-файла

Вышеприведенный пример создает новый CSV-файл и начинает записывать данные с начала. Но хотелось бы добавлять данные в существующий файл CSV вместо того, чтобы создавать новый файл.

Этого можно достичь, передав второй аргумент **boolean** экземпляру **FileWriter**:

```
CSVWriter writer = new CSVWriter(new FileWriter("staff.csv", true));
```

Тогда метод **writeNext()** дозапишет строчку в уже существующий файл:

```
String[] employee = "2,David,Feezor,USA,40".split(",");
writer.writeNext(employee);
```

Содержимое файла изменится на:

```
[1, David, Miller, Australia, 30]
[2, David, Feezor, USA, 40]
```

Пользовательский разделитель

Не всегда приходится работать с файлами, которые используют стандартный разделитель. Например, что делать, если необходимо прочитать файл с следующим содержимым:

```
4|David|Miller|Australia|30
3|David|Feezor|USA|40
```

Для чтения такого файла, необходимо создать пользовательский парсер **CSVParser**:

```
// Создадим пользовательский парсер
CSVParser parser = new CSVParserBuilder()
        .withSeparator('|')
        .build();
```

Чтобы использовать пользовательский парсер при чтении файла, необходимо создать считывать через класс **CVSReaderBuilder**:

```
// Создадим считыватель через билдер
CSVReader reader = new CSVReaderBuilder(new FileReader("staff.csv"))
        .withCSVParser(parser)
        .build();
```

Связывание CSV с Java классом

OpenCSV также предоставляет функциональные возможности для сопоставления CSV-файла с объектом Java.

Свяжем данные из файла **staff.csv** с объектом класса **Employee**:

staff.csv

1,Lokesh,Gupta,India,32
2,David,Miller,England,34

```
public class Employee  {
    private long id;
    private String firstName;
    private String lastName;
    private String country;
    private int age;

    @Override
    public String toString() {
        return "Employee{" +
                "id='" + id + '\'' +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", country='" + country + '\'' +
                ", age='" + age + '\'' +
                '}';
    }
}
```

Чтение Java класса из CSV

ColumnPositionMappingStrategy определяет класс, к которому будут привязывать данные из CSV документа, а также порядок расположения полей в этом документе:

```
ColumnPositionMappingStrategy<Employee> strategy =  new ColumnPositionMappingStrategy<>();
strategy.setType(Employee. class);
strategy.setColumnMapping( "id", "firstName", "lastName", "country", "age");
```

CsvToBean создает инструмент для взаимодействия CSV документа и выбранной ранее стратегии:

```
CsvToBean<Employee> csv = new CsvToBeanBuilder<Employee>(csvReader)
       .withMappingStrategy(strategy)
       .build();
```

CsvToBean позволяет распарсить CSV файл в список объектов, который далее можно использовать в своих целях:

```
List<Employee> list = csv.parse();
list.forEach(System.out::println);
```

Чтение Java класса из CSV

Полный код чтения CSV документа в Java класс приведен ниже:

```
try (CSVReader csvReader = new CSVReader(new FileReader("staff.csv"))) {
   ColumnPositionMappingStrategy<Employee> strategy =
          new ColumnPositionMappingStrategy<>();
   strategy.setType(Employee.class);
   strategy.setColumnMapping("id", "firstName", "lastName", "country", "age");

   CsvToBean<Employee> csv = new CsvToBeanBuilder<Employee>(csvReader)
           .withMappingStrategy(strategy)
           .build();

   List<Employee> staff = csv.parse();
   staff.forEach(System.out::println);
} catch (IOException e) {
   e.printStackTrace();
}
```

Вывод консоли:

```
Employee{id='1', firstName='Lokesh', lastName='Gupta', country='India', age='32'}
Employee{id='2', firstName='David', lastName='Miller', country='England', age='34'}
```

Запись Java класса в CSV

И наконец, стоит рассмотреть способ записи Java объекта в CSV файл.

Предварительно создадим список объектов для записи:

```
List<Employee> staff = new ArrayList<>();
staff.add(new Employee("1","Nikita", "Shumskii", "Russia", "25"));
staff.add(new Employee("2","Pavel", "Shramko", "Russia", "23"));
```

Произведем запись объектов в файл с помощью метода **write()**:

```
try(Writer writer  = new FileWriter("staff.csv")) {
   StatefulBeanToCsv<Employee> sbc =
           new StatefulBeanToCsvBuilder<Employee>(writer).build();
   sbc.write(staff);
} catch (IOException e) {
   e.printStackTrace();
} catch (CsvRequiredFieldEmptyException | CsvDataTypeMismatchException e) {
   e.printStackTrace();
}
```

Содержимое файла **“staff.csv”** после записи:

```
"AGE","COUNTRY","FIRSTNAME","ID","LASTNAME"
"25","Russia","Nikita","1","Shumskii"
"23","Russia","Pavel","2","Shramko"
```

Запись Java класса в CSV

Если необходимо записать файл с определенной последовательностью колонок, необходимо создать соответствующую стратегию:

```
List<Employee> staff =  new ArrayList<>();
staff.add(new Employee(1,"Nikita", "Shumskii", "Russia", 25));
staff.add(new Employee(2,"Pavel", "Shramko", "Russia", 23));

ColumnPositionMappingStrategy<Employee> strategy =
          new ColumnPositionMappingStrategy<>();
strategy.setType(Employee. class);
strategy.setColumnMapping( "id", "firstName", "lastName", "country", "age");

try(Writer writer  = new FileWriter("staff.csv")) {
   StatefulBeanToCsv<Employee> sbc =
           new StatefulBeanToCsvBuilder<Employee>(writer)
                   .withMappingStrategy(strategy)
                   .build();
   sbc.write(staff);
} catch (IOException | CsvRequiredFieldEmptyException | CsvDataTypeMismatchException e) {
   e.printStackTrace();
}
```

Содержимое файла **“staff.csv”** после записи:

```
"1","Nikita","Shumskii","Russia","25"
"2","Pavel","Shramko","Russia","23"
```

Контрольные вопросы

- Что такое CSV?
- Для чего он предназначен?
- Что такое разделитель?
- Для чего нужно связывать CSV с объектами Java?

Работа с XML

Что такое XML файлы

XML (eXtensible Markup Language) – это текстовое представление данных с помощью тегов в виде дерева данных.

XML файлы чаще всего используются для хранения и передачи данных. Структура такого документа хорошо описывает бизнес-объекты, конфигурацию, структуры данных и т.п.

XML представляет собой набор тегов — **узлов**. Каждый узел может иметь неограниченное количество дочерних узлов.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
   <company>
       <staff>
           <employee firstname="Иван" id="1" lastname="Иванов"/>
           <employee firstname="Петр" id="2" lastname="Петров">
               <tool>124562</tool>
           </employee>
       </staff>
       <equipment>
           <tool id="123098">Отвертка</tool>
       </equipment>
   </company>
</root>
```

Правила создания XML файлов

В общем случае XML файлы должны удовлетворять следующим требованиям:

- В заголовке файла помещается объявление XML с указанием языка разметки, номера версии и дополнительной информации.
- Каждый открывающий тэг обязательно должен иметь своего закрывающего "напарника".
- Учитывается регистр символов.
- Все значения атрибутов тэгов должны быть заключены в кавычки.
- Вложенность тэгов в XML строго контролируется.
- Учитываются все символы форматирования.

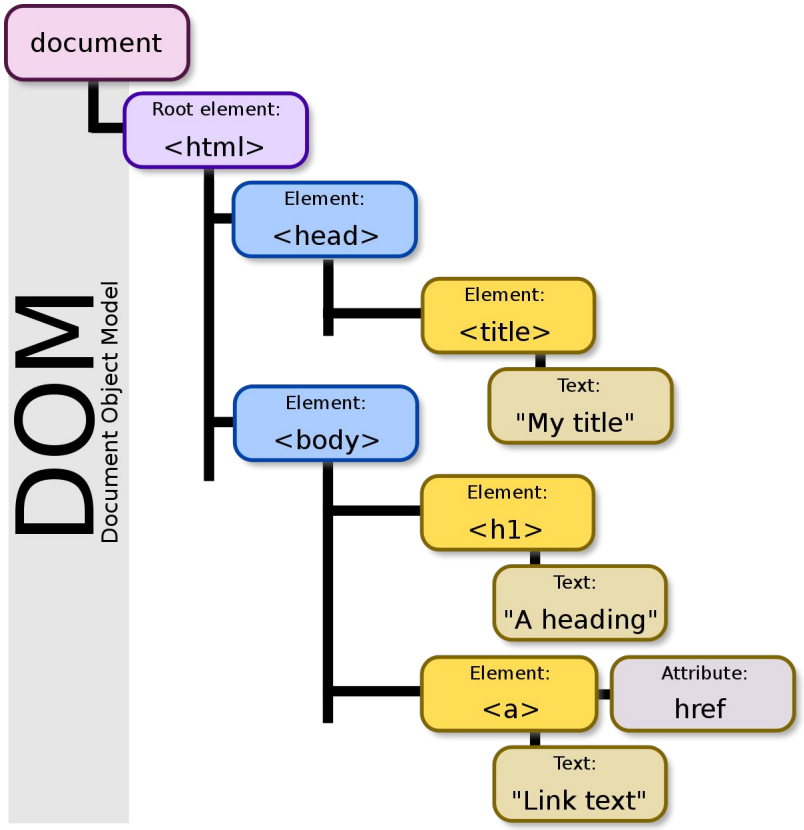
Объектная модель документа XML

Для работы с XML документами будем использовать стандартную библиотеку, входящую в JDK, а именно DOM — Document Object Model.

Так что же такое DOM? Судя из названия это есть объектная модель документа.

DOM представляет собой всё это дерево в виде специальных объектов Node. Каждый Node соответствует своему XML-тегу.

Таким образом, получается некое дерево. На самой вершине этой иерархии находится Document.



Парсинг XML файла

DOM XML парсер читает содержимое XML файла и загружает его в оперативную память. Таким образом, строится объектная модель исходного XML документа, используя которую можно работать с данными:

- читать/добавлять/удалять элементы документа
- совершать обход дерева элементов
- и другие действия

Для того, чтобы получить объект Document для нашего XML-файла необходимо выполнить следующий код.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(new File("company.xml"));
```

Получение списка узлов XML

Теперь при помощи DOM методов можно произвести разбор документа и добраться до нужного узла иерархии,чтобы прочитать его свойства.

Получим корневой узел документа:

```
Node root = doc.getDocumentElement();
System.out.println("Корневой элемент: " + root.getNodeName());
```

Получить список дочерних узлов можно при помощи метода **getChildNodes()**:

```
NodeList nodeList = root.getChildNodes();
```

Пробежаться по всем дочерним узлам текущего узла можно с помощью цикла:

```
for (int i = 0; i < nodeList.getLength(); i++) {
   Node node = nodeList.item(i);
   System.out.println("Текущий элемент: " + node.getNodeName());
}
```

Чтение содержимого элементов XML

Используя метод **getNodeType()** можно узнать тип узла:

```
if (Node.ELEMENT_NODE == node.getNodeType()) {
     Element employee = (Element) node;
     // работа с элементом
}
```

С помощью метода **getAttributes()** получается **NamedNodeMap**, который содержит атрибуты узла:

```
System.out.println("ID сотрудника: " + employee.getAttribute("id"));
```

Узлы можно найти по их тагу с помощью метода **getElementsByTagName()**:

```
System.out.println("Id инструмента: " +
employee.getElementsByTagName( "tool").item(0).getTextContent());
```

Чтение XML файла

Итоговый код для чтения документа:

```
DocumentBuilderFactory factory = DocumentBuilderFactory. newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse( new File("company.xml"));

Node root = doc.getDocumentElement();
System.out.println("Корневой элемент: " + root.getNodeName());
read(root);
```

Вывод консоли:

```
Корневой элемент: root
Текущий узел: company
Текущий узел: staff
Текущий узел: employee
Атрибут: firstname; значение: Иван
Атрибут: id; значение: 1
Атрибут: lastname; значение: Иванов
Текущий узел: employee
Атрибут: firstname; значение: Петр
Атрибут: id; значение: 2
Атрибут: lastname; значение: Петров
Текущий узел: tool
Текущий узел: equipment
Текущий узел: tool
Атрибут: id; значение: 123098
```

Используем рекурсию для чтения документа:

```
private static void read(Node node) {
    NodeList nodeList = node.getChildNodes();
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node node_ = nodeList.item(i);
        if (Node.ELEMENT_NODE == node_.getNodeType()) {
            System.out.println("Текущий узел: " + node_.getNodeName());
            Element element = (Element) node_;
            NamedNodeMap map = element.getAttributes();
            for (int a = 0; a < map.getLength(); a++) {
                String attrName = map.item(a).getNodeName();
                String attrValue = map.item(a).getNodeValue();
                System.out.println("Атрибут: " + attrName + "; значение: " + attrValue);
            }
            read(node_);
        }
    }
}
```

Создание XML документа

Для того, чтобы создать документ **Document**, необходимо воспользоваться классами **DocumentBuilderFactory** для создания билдера документа **DocumentBuilder**, который и создаст документ:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.newDocument();
```

Создадим корневой элемент документа, используя метод **appendChild()**:

```
Element root = document.createElement("root");
document.appendChild(root);
```

Добавим несколько вложенных узлов:

```
Element company = document.createElement("company");
root.appendChild(company);

Element equipment = document.createElement("equipment");
company.appendChild(equipment);

Element staff = document.createElement("staff");
company.appendChild(staff);
```

Добавление узлов в XML файл

Добавим к корневому узлу еще несколько узлов, расположенных ниже по иерархии:

```
Element employee = document.createElement("employee");
employee.setAttribute("id", "3");
employee.setAttribute("firstname", "Nikita");
employee.setAttribute("lastname", "Shumskii");
staff.appendChild(employee);
```

К добавленному узлу можем добавить еще несколько дочерних:

```
Element tool = document.createElement("tool");
tool.appendChild(document.createTextNode("123456"));
employee.appendChild(tool);
```

Запись XML документа в файл

Для записи XML документа в файл необходимо проделать три операции.

Во-первых, необходимо создать объект типа **DOMSource**, который исходя из названия, будет служить источником данных:

```
DOMSource domSource = new DOMSource(document);
```

Далее необходимо создать поток **StreamResult**, в который будет производиться запись данных из документа:

```
StreamResult streamResult = new StreamResult(new File("new_company.xml"));
```

Наконец, необходимо создать трансформер **Transformer**, который произведет преобразование документа в формат пригодный для записи в поток

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.transform(domSource, streamResult);
```

Создание и запись XML документа в файл

Полный код создания и записи в файл XML документа:

```
DocumentBuilderFactory factory = DocumentBuilderFactory. newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.newDocument();

Element root = document.createElement( "root");
document.appendChild(root);
Element company = document.createElement( "company");
root.appendChild(company);
Element equipment = document.createElement( "equipment");
company.appendChild(equipment);
Element staff = document.createElement( "staff");
company.appendChild(staff);

Element employee = document.createElement( "employee");
employee.setAttribute( "id", "3");
employee.setAttribute( "firstname", "Nikita");
employee.setAttribute( "lastname", "Shumskii");
staff.appendChild(employee);
Element tool = document.createElement( "tool");
tool.appendChild(document.createTextNode( "123456"));
employee.appendChild(tool);

DOMSource domSource = new DOMSource(document);
StreamResult streamResult = new StreamResult(new File("new_company.xml"));
TransformerFactory transformerFactory = TransformerFactory. newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.transform(domSource, streamResult);
```

Содержимое файла:

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<root>
  <company>
    <equipment/>
    <staff>
      <employee firstname="Nikita" id="3"
lastname="Shumskii">
        <tool>123456</tool>
      </employee>
    </staff>
  </company>
</root>
```

Редактирование XML документа

А что если требуется отредактировать существующий документ? Не проблема. Необходимо считать его и внести правки с помощью метода **setTextContent():**

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(new File("company.xml"));

NodeList nodeList = doc.getElementsByTagName("employee");
for (int i = 0; i < nodeList.getLength(); i++) {
   Node node = nodeList.item(i);
   if (Node.ELEMENT_NODE == node.getNodeType()) {
       Element element = (Element) node;
       if (Node.ELEMENT_NODE == node.getNodeType()) {
          Element element = (Element) node;
          if (element.getAttribute("id").equals("2")) {
               element.getElementsByTagName("tool").item(0).setTextContent("124562");
          }
       }
   }
}

DOMSource domSource = new DOMSource(doc);
StreamResult streamResult = new StreamResult(new File("company.xml"));
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.transform(domSource, streamResult);
```

Контрольные вопросы

- Что такое XML документы?
- Для каких целей используют XML файлы?
- Какие правила существуют при создании XML файла?
- Что такое объектная модель документа?
- Что такое узел?

Работа с JSON

Что такое JSON

JSON (JavaScript Object Notation) — текстовый формат обмена данными.

JSON – это формат, который хранит структурированную информацию и в основном используется для передачи данных между сервером и клиентом.

Файл JSON представляет собой более простую и лёгкую альтернативу расширению с аналогичными функциями XML (Extensive Markup Language).

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": 101101
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
 }
```

JSON Синтаксис

Чтобы правильно создать файл .json, необходимо следовать правильному синтаксису.

Есть два основных элемента объекта JSON: ключи и значения.

- Ключи должны быть строками - последовательность символов, заключенных в кавычки.
- Значения являются допустимым типом данных JSON. массив, объект, строка, логическое значение, число или значение null.

Объект JSON начинается и заканчивается фигурными скобками **{ }**. Внутри может быть две или больше пар ключей/значений с запятой для их разделения. Между тем за каждым ключом следует двоеточие, чтобы отличить его от значения.

Вот пример:

```
{"city":"New York", "country":"United States "}
```

Здесь две пары ключей/значений: ключи – город и страна; Нью-Йорк и США – значения.

Json Simple

Для работы с JSON файлами существует большое количество различных библиотек. Все они примерно похожи по своим возможностям.

В качестве примера, ознакомимся с библиотекой **Json Simple**, которая предоставляет возможность парсинга существующего JSON объекта и создание нового.

Зависимость в Maven:

```
<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
<dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
</dependency>
```

Зависимость в Gradle:

```
compile 'com.googlecode.json-simple:json-simple:1.1.1'
```

Обзор Json Simple API

Json Simple — представляет собой простой API для обработки JSON. Сам API состоит из около 13 классов, основу которых составляют следующие 5 классов:

- Класс **JSONParser** предназначен для разбора строки с JSON-содержимым. Он принимает объект java.io.Reader или строку.
- Класс **JSONObject** — это Java представление JSON строки. Класс JSONObject наследует HashMap и хранит пары ключ — значение.
- Класс **JSONArray** представляет собой коллекцию. Он наследует ArrayList и реализует интерфейсы JSONAware и JSONStreamAware.
- **JSONValue** — класс для парсинга JSON строки в Java объекты. Для этого он использует класс JSONParser. J
- Интерфейс **JSONAware**. Класс должен реализовывать этот интерфейс, чтобы конвертироваться в JSON формат.

Чтение JSON файла

Для чтения из JSON файла необходимо с помощью **JSONParser** считать файл в **Object** и привести его к типу **JSONObject**:

```
JSONParser parser = new JSONParser();
try {
   Object obj = parser.parse(new FileReader("data.json"));
   JSONObject jsonObject = (JSONObject) obj;
   System.out.println(jsonObject);
} catch (IOException | ParseException e) {
   e.printStackTrace();
}
```

data.json

```
{
 "firstName": "Иван",
 "lastName": "Иванов",
 "address": {
   "streetAddress": "Московское
ш., 101, кв.101",
   "city": "Ленинград",
   "postalCode": 101101
 },
 "phoneNumbers": [
   "812 123-1234",
   "916 123-4567"
 ]
}
```

После чтения в консоль будет выведено:

```
{"firstName":"Иван","lastName":"Иванов","address":{"streetAddress":"
Московское ш., 101, кв.101","city":"
Ленинград","postalCode":101101},"phoneNumbers":["812 123-1234","916
123-4567"]}
```

В данном примере JSON файл был считан как есть в одну строку

Разбор JSONObject

Из полученного объекта **JSONObject** можно выборочно извлечь информацию о его содержимом. Например, получим фамилию:

```
String lastName = (String) jsonObject.get("lastName");
System.out.println(lastName);
```

Из **JSONObject** можно извлечь подобъект. Например, подобъекты адреса:

```
JSONObject adress = (JSONObject) jsonObject.get("address");
String city = (String) adress.get("streetAddress");
System.out.println(city);
```

Из **JSONObject** можно извлечь массив. Например, массив с номерами телефонов:

```
JSONArray phoneNumbers = (JSONArray) jsonObject.get("phoneNumbers");
for (Object number : phoneNumbers) {
   System.out.println(number);
}
```

Вывод консоли:

```
Иванов
Московское ш., 101, кв.101
812 123-1234
916 123-4567
```

Запись JSON в файл

Для записи JSON в файл в первую очередь необходимо создать **JSONObject** и добавить в него компоненты с помощью метода **put()**:

```
JSONObject obj = new JSONObject();
obj.put("name", "mkyong.com");
obj.put("age", 100);

JSONArray list = new JSONArray();
list.add("msg 1");
list.add("msg 2");
list.add("msg 3");
obj.put("messages", list);
```

После этого произведем запись файла, используя **FileWriter**, преобразовав **JSONObject** в текст с помощью метода **toJSONString()**:

```
try (FileWriter file = new
FileWriter("new_data.json")) {
   file.write(obj.toJSONString());
   file.flush();
} catch (IOException e) {
   e.printStackTrace();
}
```

Записанный файл:

new_data.json

```
{
 "name": "mkyong.com",
 "messages": [
   "msg 1",
   "msg 2",
   "msg 3"
 ],
 "age": 100
}
```

Библиотека GSON

Библиотека GSON была разработана программистами Google и позволяет конвертировать объекты JSON в Java-объекты и наоборот.

Установим зависимость в Maven:

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
```

Установим зависимость в Gradle:

```
implementation 'com.google.code.gson:gson:2.8.2'
```

Конвертируем объект в JSON

Создадим класс Cat:

```
class Cat {
   public String name;
   public int age;
   public int color;
}
```

Создадим экземпляр класса Cat:

```
Cat cat = new Cat();
cat.name = "Матроскин";
cat.age = 5;
cat.color = Color.blue.getRGB();
```

Конвертируем объект созданного класса в JSON при помощи метода **toJson()**:

```
GsonBuilder builder = new GsonBuilder();
Gson gson = builder.create();
System.out.println(gson.toJson(cat));
```

В логах видим строку:

```
{"name":"Матроскин","age":5,"color":-16776961}
```

Конвертируем JSON в объект

Естественно, требуется и производить конвертацию из JSON в Java класс.

Допустим с сервера пришёл ответ в виде JSON-строки и требуется из нее построить объект:

```
String jsonText = "{\"name\":\"Мурзик\",\"color\":-16777216,\"age\":9}";
```

В этом случае вызывается метод **fromJson()**:

```
GsonBuilder builder = new GsonBuilder();
Gson gson = builder.create();
Cat cat = gson.fromJson(jsonText, Cat.class);
System.out.println("Имя: " + cat.name + "\nВозраст: " + cat.age + "\nЦвет: " +
cat.color);
```

Вывод консоли:

```
Имя: Мурзик
Возраст: 9
Цвет: -16777216
```

Контрольные вопросы

- Что такое JSON?
- С чем можно сравнить синтаксис JSON?
- Какие объекты можно добавлять в JSON?

Итоги

Итоги

- Файлы CSV содержат значения, разделенные запятыми, и используются для экспорта и импорта данных.
- JDK не имеет инструментов для работы с файлами CSV, поэтому используются сторонние библиотеки.
- OpenCSV — библиотека парсеров CSV файлов, поддерживающая весь основной функционал для работы с CSV.

Итоги

- Файлы XML это текстовое представление данных с помощью тегов в виде дерева данных, используемые для хранения и передачи данных.
- Для работы с XML документами может быть использована стандартная библиотеку JDK, а именно DOM — Document Object Model.
- DOM позволяет произвести разбор документа и добраться до нужного узла иерархии и прочитать его свойства.

Итоги

- JSON (JavaScript Object Notation) — структурированный текстовый формат обмена данными между сервером и клиентом.
- Библиотека Json Simple представляет собой простой API для обработки JSON.
- Библиотека GSON позволяет конвертировать объекты JSON в Java-объекты и наоборот.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Григорий Вахмистров