16. XML与JSON

16.1. 什么是XML

- XML（Extensible Markup Language可扩展标记语言），XML是一个以文本来描述数据的文档

16.2. XML的用途

- 用途：
  - 充当显示数据（以XML充当显示层）
  - 存储数据（存储层）的功能
  - 以XML描述数据，并在联系服务器与系统的其余部分之间传递（传输数据的一种格式）
- 从某种角度来讲，XML是数据封装和消息传递技术

16.3. SAX解析XML

- SAX是Simple API for XML的缩写
- SAX是读取和操作XML数据更快速、更轻量的方法。SAX允许您在读取文档时处理它，从而不必等待整个文档被存储之后才采取操作。它不涉及DOM所必须的开销和概念跳跃。SAX API是一个基于事件的API，适用于处理数据流，即随着数据的流动而依次处理数据。SAX API在其解析您的文档时发生一定事件的时候会通知您。在您对其响应时，您不保存的数据将会被抛弃
- SAX API中主要有四种处理事件的接口，它们分别是ContentHandler，DTDHandler，EntityResolver和ErrorHandler。实际上只要继承DefaultHandller类就可以，DefaultHandler实现了这四个事件处理器接口，然后提供了每个抽象方法的默认实现

```
/**
 * SAX解析的特点:
 * 1.基于事件驱动
 * 2.顺序读取，速度快
```

```
 * 3.不能任意读取节点（灵活性差）
 * 4.解析时占用的内存小
 * 5.SAX更适用于在性能要求更高的设备上使用（Android开发中）
 *
 * @throws ParserConfigurationException
 * @throws SAXException
 * @throws IOException
 */
@Test
public void saxParseXML() throws ParserConfigurationException,
SAXException, IOException {
    // 1.创建一个SAX解析器工厂对象
    SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
    // 2.通过工厂对象创建一个SAX解析器
    SAXParser saxParser = saxParserFactory.newSAXParser();
    // 3.创建一个数据处理器（需要我们自己来编写）
    PersonHandler personHandler = new PersonHandler();
    // 4.开始解析
    InputStream is = Thread.currentThread().getContextClassLoader()
                .getResourceAsStream("day16_XML与JSON/xml/person.xml");
    saxParser.parse(is, personHandler);
    List<Person> persons = personHandler.getPersons();
    for (Person p : persons) {
        System.out.println(p);
    }
}
```

16.4. DOM解析XML

- JAVA解析XML通常有两种方式，DOM和SAX
- DOM：Document Object Model（文档对象模型）
- DOM的特性：定义一组Java接口，基于对象，与语言和平台无关将XML文档表示为树，在内存中解析和存储XML文档，允许随机访问文档的不同部分
- DOM的优点：由于树在内存中是持久的，因此可以修改后更新。它还可以在任何时候在树中上下导航，API使用起来也比较简单
- 解析步骤：

```
/**
 * DOM解析XML
 * 1.基于属性结构，通过解析器一次性把文档加载到内存中，所以比较占用内存，可以随机访问，更加灵活，更适合在web开发中使用
 * @throws ParserConfigurationException
 * @throws IOException
 * @throws SAXException
 */
@Test
public void domParseXML() throws ParserConfigurationException,
IOException, SAXException {
    // 1.创建一个DOM解析器工厂对象
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    // 2.通过工厂对象生成解析器对象
    DocumentBuilder documentBuilder = factory.newDocumentBuilder();
```

```
    // 3.解析文档
    InputStream is = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream("day16_XML与JSON/xml/person.xml");
    // 此代码完成后， 整个XML文档已经被加载到内存中，以树状形式存储
    Document doc = documentBuilder.parse(is);
    // 4.从内存中读取数据
    // 获取节点名称为person的所有节点，返回节点集合
    NodeList personNodeList = doc.getElementsByTagName("person");
    ArrayList<Person> persons = new ArrayList<>();
    Person person = null;
    // 此循环迭代两次
    for (int i = 0; i < personNodeList.getLength(); i++) {
        Node personNode = personNodeList.item(i);
        person = new Person();
        // 获取节点的属性值
        String personid =
personNode.getAttributes().getNamedItem("personid").getNodeValue();
        person.setPersonId(personid);
        // 获取当前节点的所有子节点
        NodeList childNodes = personNode.getChildNodes();
        for (int j = 0; j < childNodes.getLength(); j++) {
            Node item = childNodes.item(j);
            String nodeName = item.getNodeName();
            if ("name".equals(nodeName)) {
                person.setName(item.getFirstChild().getNodeValue());
            } else if ("address".equals(nodeName)) {
                person.setAddress(item.getFirstChild().getNodeValue());
            } else if ("tel".equals(nodeName)) {
                person.setTel(item.getFirstChild().getNodeValue());
            } else if ("fax".equals(nodeName)) {
                person.setFax(item.getFirstChild().getNodeValue());
            } else if ("email".equals(nodeName)) {
                person.setEmail(item.getFirstChild().getNodeValue());
            }
        }
        persons.add(person);
    }
    System.out.println("结果: ");
    System.out.println(Arrays.toString(persons.toArray()));
}
```

16.5. JDOM解析XML

- JDOM简化了与XML的交互作用并且比使用DOM实现更快，JDOM和DOM主要也有两方面的不同。首先JDOM仅使用具体类而不使用接口。这在某些方面简化了API，但是也限制了灵活性。第二，API大量使用了Collections类，简化了那些已经熟悉这些类的Java开发者的使用
- 下载地址：http://www.jdom.org/downloads/index.html
- 解析步骤：

```
/**
 * JDOM解析XML
 * 1.与DOM类似基于树形结构
 * 2.与DOM的区别:
```

```
 * （1）第三方开源的组件
 * （2）实现使用JAVA的Collections接口
 * （3）效率比DOM更快
 * @throws JDOMException
 * @throws IOException
 */
@Test
public void jdomParseXML() throws JDOMException, IOException {
    // 创建JDOM解析器
    SAXBuilder builder = new SAXBuilder();
    InputStream is = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream("day16_XML与JSON/xml/person.xml");
    org.jdom2.Document build = builder.build(is);
    Element rootElement = build.getRootElement();
    List<Person> list = new ArrayList<>();
    Person person = null;
    List<Element> children = rootElement.getChildren();
    for (Element child : children) {
        person = new Person();
        String personid = child.getAttributeValue("personid");
        person.setPersonId(personid);
        List<Element> children1 = child.getChildren();
        for (Element element : children1) {
            String name = element.getName();
            if ("name".equals(name)) {
                person.setName(element.getText());
            } else if ("address".equals(name)) {
                person.setAddress(element.getText());
            } else if ("tel".equals(name)) {
                person.setTel(element.getText());
            } else if ("fax".equals(name)) {
                person.setFax(element.getText());
            } else if ("email".equals(name)) {
                person.setEmail(element.getText());
            }
        }
        list.add(person);
    }
    System.out.println("结果: ");
    System.out.println(Arrays.toString(list.toArray()));
}
```

16.6. DOM4J解析XML

- dom4j是一个非常优秀的Java XML API，具有性能优异、功能强大和极端易用使用的特点，同时它也是一个开放源代码的软件，可以在SourceForge上找到它。在对主流的Java XML API进行的性能、功能和易用的评测，dom4j无论在那个方面都是非常出色的。Sun的JAXM也在用dom4j，Hibernate用它来读写配置文件
- 下载地址：https://dom4j.github.io/
- 解析步骤：

```
/**
 * DOM4J解析XML
```

```
 * 1.基于树形结构，第三方组件
 * 2.解析速度快，效率更高，使用Java中的迭代器实现数据读取，在web框架中使用较多（Hibernate）
 *
 * @throws DocumentException
 */
@Test
public void dom4jParseXML() throws DocumentException {
    // 1.创建DOM4J的解析器结果
    SAXReader reader = new SAXReader();
    InputStream is = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream("day16_XML与JSON/xml/person.xml");
    org.dom4j.Document doc = reader.read(is);
    org.dom4j.Element rootElement = doc.getRootElement();
    Iterator<org.dom4j.Element> iterator = rootElement.elementIterator();
    List<Person> persons = new ArrayList<>();
    Person person = null;
    while (iterator.hasNext()) {
        person = new Person();
        org.dom4j.Element element = iterator.next();
        String personid = element.attributeValue("personid");
        person.setPersonId(personid);
        Iterator<org.dom4j.Element> iterator1 = element.elementIterator();
        while (iterator1.hasNext()) {
            org.dom4j.Element next = iterator1.next();
            String name = next.getName();
            if ("name".equals(name)) {
                person.setName(next.getText());
            } else if ("address".equals(name)) {
                person.setAddress(next.getText());
            } else if ("tel".equals(name)) {
                person.setTel(next.getText());
            } else if ("fax".equals(name)) {
                person.setFax(next.getText());
            } else if ("email".equals(name)) {
                person.setEmail(next.getText());
            }
        }
        persons.add(person);
    }
    System.out.println("结果: ");
    System.out.println(Arrays.toString(persons.toArray()));
}
```

16.7. 通过对象生成XML文件

- 根据对象生成XML文档
- 使用Java提供的java.beans.XMLEncoder和java.beans.XMLDecoder类
- 步骤:

```
// 写入
/**
 * 把对象写入XML文件
```

```
 * @throws FileNotFoundException
 */
@Test
public void xmlEnCoder() throws FileNotFoundException {
    BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("src/day16_XML与JSON/xml/test.xml"));
    XMLEncoder xmlEncoder = new XMLEncoder(bos);
    Person person = new Person();
    person.setPersonId("1212");
    person.setName("89");
    person.setAddress("北京");
    person.setTel("13838389723");
    person.setFax("6768789798");
    person.setEmail("vince@163.com");
    xmlEncoder.writeObject(person);
    xmlEncoder.close();
}

// 读取
/**
 * 从XML文件中读取对象
 * @throws FileNotFoundException
 */
@Test
public void xmlDeCoder() throws FileNotFoundException {
    BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("src/day16_XML与JSON/xml/test.xml"));
    XMLDecoder xmlDecoder = new XMLDecoder(bis);
    Person person = (Person) xmlDecoder.readObject();
    System.out.println(person);
}
```

- xstream

```
/**
 * 使用第三方xstream组件实现XML的解析与生成
 */
@Test
public void xStream() {
    Person person = new Person();
    person.setPersonId("123");
    person.setName("812");
    person.setAddress("西安");
    person.setTel("320392094");
    person.setFax("3348092844");
    person.setEmail("faf@163.com");
    XStream xStream = new XStream(new Xpp3DomDriver());
    xStream.alias("person", Person.class);
    xStream.useAttributeFor(Person.class, "personId");
    String xml = xStream.toXML(person);
    System.out.println(xml);

    // 解析
    Person p = (Person) xStream.fromXML(xml);
    System.out.println(p);
}
```

16.8. 各种解析方法比较

- JDOM和DOM在性能测试时表现不佳，在测试10M文档时内存溢出
- SAX表现较好，这要依赖于它特定的解析方式。一个SAX检测即将到来的XML流，但并没有载入到内存（当然XML流被读入时，会有部分文档暂时隐藏在内存中）
- 很多开源项目中使用了DOM4J，Hibernate也用DOM4J来读取XML配置文件
- xstream实现XML的转换

16.9. JSON

- JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。JSON官方：https://www.json.org/
- JSON数据格式的特点：
  - JSON建构于两种结构：
    - “名称/值”对的集合
    - 值的有序列表（数组）
  - JSON表示名称/值对的方式：

```
{
    "firstName" : "vince",
    "lastName" : "ma",
    "email" : "finally_m@foxmail.com"
}
```

  - 表示数组：

```
{
    "user" : [{
        "firstName" : "vince",
        "lastName" : "ma",
        "email" : "finally_m@foxmail.com"
    }, {
        "firstName" : "lin",
        "lastName" : "jacks",
        "email" : "jacks@qq.com"
    }]
}
```

- JsonReader解析简单JSON数据：

```
import com.google.gson.stream.JsonReader;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;

```

```java
/**
 * @author xiao儿
 * @date 2019/9/14 17:56
 * @Description JSONDemo
 */
public class JSONDemo {
    /**
     * 解析一个JSON数组
     */
    @Test
    public void parseJSONNames() {
        InputStream is = Thread.currentThread().getContextClassLoader()
                .getResourceAsStream("day16_XML与JSON/json/names.json");
        InputStreamReader in = new InputStreamReader(is);
        // JSON的解析工具（解析器）
        JsonReader reader = new JsonReader(in);
        ArrayList<Name> list = new ArrayList<>();
        try {
            // 开始解析数组
            reader.beginArray();
            while (reader.hasNext()) {
                list.add(parseName(reader));
            }
            // 结束解析数组
            reader.endArray();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(Arrays.toString(list.toArray()));
    }

    // 解析对象 Name
    private Name parseName(JsonReader jsonReader) {
        Name name = null;
        try {
            // 开始解析对象
            jsonReader.beginObject();
            name = new Name();
            while (jsonReader.hasNext()) {
                String attrName = jsonReader.nextName();
                if ("firstName".equals(attrName)) {
                    name.setFirstName(jsonReader.nextString());
                } else if ("lastName".equals(attrName)) {
                    name.setLastName(jsonReader.nextString());
                } else if ("email".equals(attrName)) {
                    name.setEmail(jsonReader.nextString());
                }
            }
            // 结束解析对象
            jsonReader.endObject();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return name;
    }
}
```

- JsonReader解析复杂JSON数据：

```
import com.google.gson.stream.JsonReader;
import com.google.gson.stream.JsonToken;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

/**
 * @author xiao儿
 * @date 2019/9/14 19:22
 * @Description JSONMessage
 */
public class JSONMessage {
    /**
     * 使用JsonReader解析复杂数据
     */
    @Test
    public void parseJSONMessage() {
        InputStream is =
Thread.currentThread().getContextClassLoader()
                .getResourceAsStream("day16_XML与
JSON/json/message.json");
        InputStreamReader in = new InputStreamReader(is);
        JsonReader reader = new JsonReader(in);
        ArrayList<Message> list = readMessageArray(reader);
        for (Message message : list) {
            System.out.println(message);
        }
    }

    /**
     * 读取Message数组
     * @param reader
     * @return
     */
    private ArrayList<Message> readMessageArray(JsonReader reader) {
        ArrayList<Message> list = new ArrayList<>();
        try {
            reader.beginArray();
            while (reader.hasNext()) {
                list.add(readerMessage(reader));
            }
            reader.endArray();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return list;
    }

    /**
     * 解析一个Message对象
     * @param reader
     * @return
```

```
     */
    private Message readerMessage(JsonReader reader) {
        Message message = new Message();
        try {
            reader.beginObject();
            while (reader.hasNext()) {
                String name = reader.nextName();
                if ("id".equals(name)) {
                    message.setId(reader.nextLong());
                } else if ("text".equals(name)) {
                    message.setText(reader.nextString());
                } else if ("geo".equals(name) && reader.peek() !=
JsonToken.NULL) {
                    message.setGeo(readGeo(reader));
                } else if ("user".equals(name)) {
                    message.setUser(readUser(reader));
                } else {
                    reader.skipValue();
                }
            }
            reader.endObject();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return message;
    }

    /**
     * 解析Geo
     * @param reader
     * @return
     */
    private ArrayList<Double> readGeo(JsonReader reader) {
        ArrayList<Double> list = new ArrayList<>();
        try {
            reader.beginArray();
            while (reader.hasNext()) {
                list.add(reader.nextDouble());
            }
            reader.endArray();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return list;
    }

    /**
     * 解析User
     * @param reader
     * @return
     */
    private User readUser(JsonReader reader) {
        User user = new User();
        try {
            reader.beginObject();
            while (reader.hasNext()) {
                String name = reader.nextName();
                if ("name".equals(name)) {
```

```
                    user.setName(reader.nextString());
                } else if ("followers_count".equals(name)) {
                    user.setFollowers_count(reader.nextInt());
                } else {
                    reader.skipValue();
                }
            }
            reader.endObject();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return user;
    }
}
```

16.10. GSON组件的使用

- GSON是Google开发的Java API，用于转换Java对象和Json对象
- 下载地址：https://mvnrepository.com/artifact/com.google.code.gson/gson
- 解析JSON：

```
JsonReader reader = new JsonReader(new StringReader(jsonData));
```

- 生成JSON：

```
/**
 * 把一个JSON对象转换成JAVA对象，或把一个JAVA对象转换成JSON对象
 */
@Test
public void createJSON() {
    List<Name> list = new ArrayList<>();
    list.add(new Name("Vince", "ma", "fsaf@33.com"));
    list.add(new Name("Bin", "fs", "12@w23.com"));
    JsonArray array = new JsonArray();
    for (Name name : list) {
        JsonObject jsonObject = new JsonObject();
        jsonObject.addProperty("firstName", name.getFirstName());
        jsonObject.addProperty("lastName", name.getLastName());
        jsonObject.addProperty("email", name.getEmail());
        array.add(jsonObject);
    }
    System.out.println(array.toString());
}
```

- GSON组件的使用：

```
/**
 * 把一组JSON对象转换成一个JAVA对象集合，或者把一个JAVA对象集合转换成JSON数组
 */
@Test
public void gson2() {
    Gson gson = new Gson();
    InputStream is = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream("day16_XML与JSON/json/names.json");
```

```
9       InputStreamReader in = new InputStreamReader(is);
10      TypeToken<List<Name>> type = new TypeToken<List<Name>>() {
11      };
12      List<Name> list = gson.fromJson(in, type.getType());
13      System.out.println(list);
14      String s = gson.toJson(list);
15      System.out.println(s);
16  }
```

16.11. XML与JSON的比较

- JSON和XML的数据可读性基本相同
- JSON和XML同样拥有丰富的解析手段
- JSON相对于XML来讲，数据的体积小
- JSON与JavaScript的交互更加方便
- JSON对数据的描述性比XML较差
- JSON的速度要远远快于XML
- 适合的场景：
    - （1）数据传输：JSON要比XML更有优势
    - （2）存储数据：XML描述性更强
    - XML通常用做配置文件