

16. XML与JSON

- 16.1. 什么是XML
- 16.2. XML的用途
- 16.3. SAX解析XML
- 16.4. DOM解析XML
- 16.5. JDOM解析XML
- 16.6. DOM4J解析XML
- 16.7. 通过对象生成XML文件
- 16.8. 各种解析方法比较
- 16.9. JSON
- 16.10. GSON组件的使用
- 16.11. XML与JSON的比较

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DefaultHandler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

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

```

5      * 3.不能任意读取节点（灵活性差）
6      * 4.解析时占用的内存小
7      * 5.SAX更适用于在性能要求更高的设备上使用（Android开发中）
8      *
9      * @throws ParserConfigurationException
10     * @throws SAXException
11     * @throws IOException
12     */
13     @Test
14     public void saxParseXML() throws ParserConfigurationException,
15         SAXException, IOException {
16         // 1.创建一个SAX解析器工厂对象
17         SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
18         // 2.通过工厂对象创建一个SAX解析器
19         SAXParser saxParser = saxParserFactory.newSAXParser();
20         // 3.创建一个数据处理器（需要我们自己来编写）
21         PersonHandler personHandler = new PersonHandler();
22         // 4.开始解析
23         InputStream is = Thread.currentThread().getContextClassLoader()
24             .getResourceAsStream("day16_XML与JSON/xml/person.xml");
25         saxParser.parse(is, personHandler);
26         List<Person> persons = personHandler.getPersons();
27         for (Person p : persons) {
28             System.out.println(p);
29         }
30     }

```

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 and 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：

```

1      /**
2      * DOM解析XML
3      * 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
4      * @throws ParserConfigurationException
5      * @throws IOException
6      * @throws SAXException
7      */
8     @Test
9     public void domParseXML() throws ParserConfigurationException,
10         IOException, SAXException {
11         // 1.创建一个DOM解析器工厂对象
12         DocumentBuilderFactory factory =
13             DocumentBuilderFactory.newInstance();
14         // 2.通过工厂对象生成解析器对象
15         DocumentBuilder documentBuilder = factory.newDocumentBuilder();

```

```

14 // 3. 解析文档
15 InputStream is = Thread.currentThread().getContextClassLoader()
16     .getResourceAsStream("day16_XML与JSON/xml/person.xml");
17 // 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
18 Document doc = documentBuilder.parse(is);
19 // 4. 从内存中读取数据
20 // 获取节点名称为person的所有节点，返回节点集合
21 NodeList personNodeList = doc.getElementsByTagName("person");
22 ArrayList<Person> persons = new ArrayList<>();
23 Person person = null;
24 // 此循环迭代两次
25 for (int i = 0; i < personNodeList.getLength(); i++) {
26     Node personNode = personNodeList.item(i);
27     person = new Person();
28     // 获取节点的属性值
29     String personid =
personNode.getAttributes().getNamedItem("personid").getNodeValue();
30     person.setPersonId(personid);
31     // 获取当前节点的所有子节点
32     NodeList childNodes = personNode.getChildNodes();
33     for (int j = 0; j < childNodes.getLength(); j++) {
34         Node item = childNodes.item(j);
35         String nodeName = item.getNodeName();
36         if ("name".equals(nodeName)) {
37             person.setName(item.getFirstChild().getNodeValue());
38         } else if ("address".equals(nodeName)) {
39             person.setAddress(item.getFirstChild().getNodeValue());
40         } else if ("tel".equals(nodeName)) {
41             person.setTel(item.getFirstChild().getNodeValue());
42         } else if ("fax".equals(nodeName)) {
43             person.setFax(item.getFirstChild().getNodeValue());
44         } else if ("email".equals(nodeName)) {
45             person.setEmail(item.getFirstChild().getNodeValue());
46         }
47     }
48     persons.add(person);
49 }
50 System.out.println("结果: ");
51 System.out.println(Arrays.toString(persons.toArray()));
52 }

```

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：

```

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

```

```

5  * (1) 第三方开源的组件
6  * (2) 实现使用JAVA的Collections接口
7  * (3) 效率比DOM更快
8  * @throws JDOMException
9  * @throws IOException
10 */
11 @Test
12 public void jdomParseXML() throws JDOMException, IOException {
13     // 创建JDOM解析器
14     SAXBuilder builder = new SAXBuilder();
15     InputStream is = Thread.currentThread().getContextClassLoader()
16         .getResourceAsStream("day16_XML与JSON/xml/person.xml");
17     org.jdom2.Document build = builder.build(is);
18     Element rootElement = build.getRootElement();
19     List<Person> list = new ArrayList<>();
20     Person person = null;
21     List<Element> children = rootElement.getChildren();
22     for (Element child : children) {
23         person = new Person();
24         String personid = child.getAttributeValue("personid");
25         person.setPersonId(personid);
26         List<Element> children1 = child.getChildren();
27         for (Element element : children1) {
28             String name = element.getName();
29             if ("name".equals(name)) {
30                 person.setName(element.getText());
31             } else if ("address".equals(name)) {
32                 person.setAddress(element.getText());
33             } else if ("tel".equals(name)) {
34                 person.setTel(element.getText());
35             } else if ("fax".equals(name)) {
36                 person.setFax(element.getText());
37             } else if ("email".equals(name)) {
38                 person.setEmail(element.getText());
39             }
40         }
41         list.add(person);
42     }
43     System.out.println("结果: ");
44     System.out.println(Arrays.toString(list.toArray()));
45 }

```

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:

```

1  /**
2  * DOM4J解析XML

```

```

3  * 1.基于树形结构,第三方组件
4  * 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)
5  *
6  * @throws DocumentException
7  */
8  @Test
9  public void dom4jParseXML() throws DocumentException {
10     // 1.创建DOM4J的解析器结果
11     SAXReader reader = new SAXReader();
12     InputStream is = Thread.currentThread().getContextClassLoader()
13         .getResourceAsStream("day16_XML与JSON/xml/person.xml");
14     org.dom4j.Document doc = reader.read(is);
15     org.dom4j.Element rootElement = doc.getRootElement();
16     Iterator<org.dom4j.Element> iterator =
17     rootElement.elementIterator();
18     List<Person> persons = new ArrayList<>();
19     Person person = null;
20     while (iterator.hasNext()) {
21         person = new Person();
22         org.dom4j.Element element = iterator.next();
23         String personid = element.attributeValue("personid");
24         person.setPersonId(personid);
25         Iterator<org.dom4j.Element> iterator1 =
26         element.elementIterator();
27         while (iterator1.hasNext()) {
28             org.dom4j.Element next = iterator1.next();
29             String name = next.getName();
30             if ("name".equals(name)) {
31                 person.setName(next.getText());
32             } else if ("address".equals(name)) {
33                 person.setAddress(next.getText());
34             } else if ("tel".equals(name)) {
35                 person.setTel(next.getText());
36             } else if ("fax".equals(name)) {
37                 person.setFax(next.getText());
38             } else if ("email".equals(name)) {
39                 person.setEmail(next.getText());
40             }
41         }
42         persons.add(person);
43     }
44     System.out.println("结果: ");
45     System.out.println(Arrays.toString(persons.toArray()));
46 }

```

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:

```

1  // 写入
2  /**
3  * 把对象写入XML文件

```

```

4      * @throws FileNotFoundException
5      */
6      @Test
7      public void xmlEncoder() throws FileNotFoundException {
8          BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("src/day16_XML与JSON/xml/test.xml"));
9          XMLEncoder xmlEncoder = new XMLEncoder(bos);
10         Person person = new Person();
11         person.setPersonId("1212");
12         person.setName("89");
13         person.setAddress("北京");
14         person.setTel("13838389723");
15         person.setFax("6768789798");
16         person.setEmail("vince@163.com");
17         xmlEncoder.writeObject(person);
18         xmlEncoder.close();
19     }
20
21     // 读取
22     /**
23      * 从XML文件中读取对象
24      * @throws FileNotFoundException
25      */
26     @Test
27     public void xmlDecoder() throws FileNotFoundException {
28         BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("src/day16_XML与JSON/xml/test.xml"));
29         XMLDecoder xmlDecoder = new XMLDecoder(bis);
30         Person person = (Person) xmlDecoder.readObject();
31         System.out.println(person);
32     }

```

- xstream

```

1      /**
2      * 使用第三方xstream组件实现XML的解析与生成
3      */
4      @Test
5      public void xStream() {
6          Person person = new Person();
7          person.setPersonId("123");
8          person.setName("812");
9          person.setAddress("西安");
10         person.setTel("320392094");
11         person.setFax("3348092844");
12         person.setEmail("faf@163.com");
13         XStream xStream = new XStream(new Xpp3DomDriver());
14         xStream.alias("person", Person.class);
15         xStream.useAttributeFor(Person.class, "personId");
16         String xml = xStream.toXML(person);
17         System.out.println(xml);
18
19         // 解析
20         Person p = (Person) xStream.fromXML(xml);
21         System.out.println(p);
22     }

```

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:

```
1 {  
2     "firstName" : "vince",  
3     "lastName" : "ma",  
4     "email" : "finally_m@foxmail.com"  
5 }
```

- 表示数组:

```
1 {  
2     "user" : [{  
3         "firstName" : "vince",  
4         "lastName" : "ma",  
5         "email" : "finally_m@foxmail.com"  
6     }, {  
7         "firstName" : "lin",  
8         "lastName" : "jacks",  
9         "email" : "jacks@qq.com"  
10    }]  
11 }
```

- JsonReader解析简单JSON数据:

```
1 import com.google.gson.stream.JsonReader;  
2 import org.junit.Test;  
3  
4 import java.io.IOException;  
5 import java.io.InputStream;  
6 import java.io.InputStreamReader;  
7 import java.util.ArrayList;  
8 import java.util.Arrays;  
9
```

```

10  /**
11   * @author xiao儿
12   * @date 2019/9/14 17:56
13   * @Description JSONDemo
14   */
15  public class JSONDemo {
16      /**
17       * 解析一个JSON数组
18       */
19      @Test
20      public void parseJSONNames() {
21          InputStream is = Thread.currentThread().getContextClassLoader()
22              .getResourceAsStream("day16_XML与JSON/json/names.json");
23          InputStreamReader in = new InputStreamReader(is);
24          // JSON的解析工具（解析器）
25          JsonReader reader = new JsonReader(in);
26          ArrayList<Name> list = new ArrayList<>();
27          try {
28              // 开始解析数组
29              reader.beginArray();
30              while (reader.hasNext()) {
31                  list.add(parseName(reader));
32              }
33              // 结束解析数组
34              reader.endArray();
35          } catch (IOException e) {
36              e.printStackTrace();
37          }
38          System.out.println(Arrays.toString(list.toArray()));
39      }
40
41      // 解析对象 Name
42      private Name parseName(JsonReader jsonReader) {
43          Name name = null;
44          try {
45              // 开始解析对象
46              jsonReader.beginObject();
47              name = new Name();
48              while (jsonReader.hasNext()) {
49                  String attrName = jsonReader洗nextName();
50                  if ("firstName".equals(attrName)) {
51                      name.setFirstName(jsonReader.nextString());
52                  } else if ("lastName".equals(attrName)) {
53                      name.setLastName(jsonReader.nextString());
54                  } else if ("email".equals(attrName)) {
55                      name.setEmail(jsonReader.nextString());
56                  }
57              }
58              // 结束解析对象
59              jsonReader.endObject();
60          } catch (IOException e) {
61              e.printStackTrace();
62          }
63          return name;
64      }
65  }

```


- JsonReader解析复杂JSON数据:

```
1  import com.google.gson.stream.JsonReader;
2  import com.google.gson.stream.JsonToken;
3  import org.junit.Test;
4
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.InputStreamReader;
8  import java.util.ArrayList;
9
10 /**
11  * @author xiao儿
12  * @date 2019/9/14 19:22
13  * @Description JSONMessage
14  */
15 public class JSONMessage {
16     /**
17      * 使用JsonReader解析复杂数据
18      */
19     @Test
20     public void parseJSONMessage() {
21         InputStream is =
22 Thread.currentThread().getContextClassLoader()
23         .getResourceAsStream("day16_XML与
24 JSON/json/message.json");
25         InputStreamReader in = new InputStreamReader(is);
26         JsonReader reader = new JsonReader(in);
27         ArrayList<Message> list = readMessageArray(reader);
28         for (Message message : list) {
29             System.out.println(message);
30         }
31     }
32
33     /**
34      * 读取Message数组
35      * @param reader
36      * @return
37      */
38     private ArrayList<Message> readMessageArray(JsonReader reader) {
39         ArrayList<Message> list = new ArrayList<>();
40         try {
41             reader.beginArray();
42             while (reader.hasNext()) {
43                 list.add(readerMessage(reader));
44             }
45             reader.endArray();
46         } catch (IOException e) {
47             e.printStackTrace();
48         }
49         return list;
50     }
51
52     /**
53      * 解析一个Message对象
54      * @param reader
55      * @return
```

```

54     */
55     private Message readerMessage(JsonReader reader) {
56         Message message = new Message();
57         try {
58             reader.beginObject();
59             while (reader.hasNext()) {
60                 String name = reader洗洗Name();
61                 if ("id".equals(name)) {
62                     message.setId(reader.nextLong());
63                 } else if ("text".equals(name)) {
64                     message.setText(reader.nextString());
65                 } else if ("geo".equals(name) && reader.peek() !=
JsonToken.NULL) {
66                     message.setGeo(readGeo(reader));
67                 } else if ("user".equals(name)) {
68                     message.setUser(readUser(reader));
69                 } else {
70                     reader.skipValue();
71                 }
72             }
73             reader.endObject();
74         } catch (IOException e) {
75             e.printStackTrace();
76         }
77         return message;
78     }
79
80     /**
81      * 解析Geo
82      * @param reader
83      * @return
84      */
85     private ArrayList<Double> readGeo(JsonReader reader) {
86         ArrayList<Double> list = new ArrayList<>();
87         try {
88             reader.beginArray();
89             while (reader.hasNext()) {
90                 list.add(reader.nextDouble());
91             }
92             reader.endArray();
93         } catch (IOException e) {
94             e.printStackTrace();
95         }
96         return list;
97     }
98
99     /**
100     * 解析User
101     * @param reader
102     * @return
103     */
104     private User readUser(JsonReader reader) {
105         User user = new User();
106         try {
107             reader.beginObject();
108             while (reader.hasNext()) {
109                 String name = reader洗洗Name();
110                 if ("name".equals(name)) {

```

```

111         user.setName(reader.nextString());
112     } else if ("followers_count".equals(name)) {
113         user.setFollowers_count(reader.nextInt());
114     } else {
115         reader.skipValue();
116     }
117 }
118 reader.endObject();
119 } catch (IOException e) {
120     e.printStackTrace();
121 }
122 return user;
123 }
124 }

```

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：

```

1 | JsonReader reader = new JsonReader(new StringReader(jsonData));

```

- 生成JSON：

```

1 | /**
2 |  * 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
3 |  */
4 | @Test
5 | public void createJSON() {
6 |     List<Name> list = new ArrayList<>();
7 |     list.add(new Name("Vince", "ma", "fsaf@33.com"));
8 |     list.add(new Name("Bin", "fs", "12@w23.com"));
9 |     JSONArray array = new JSONArray();
10 |    for (Name name : list) {
11 |        JsonObject jsonObject = new JsonObject();
12 |        jsonObject.addProperty("firstName", name.getFirstName());
13 |        jsonObject.addProperty("lastName", name.getLastName());
14 |        jsonObject.addProperty("email", name.getEmail());
15 |        array.add(jsonObject);
16 |    }
17 |    System.out.println(array.toString());
18 | }

```

- GSON组件的使用：

```

1 | /**
2 |  * 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
3 |  */
4 | @Test
5 | public void gson2() {
6 |     Gson gson = new Gson();
7 |     InputStream is = Thread.currentThread().getContextClassLoader()
8 |         .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