XML与JSON

XML

简介

```
可扩展标记语言（eXtensible Markup Language）。

特性:
    1.  xml具有平台无关性，是一门独立的标记语言.
    2.  xml具有自我描述性
```

为什么学习XML？

```
    1.  网络数据传输.
    2.  数据存储
    3.  配置文件
```

XML文件

```
.XML文件是保存XML数据的一种方式
XML数据也可以以其他的方式存在（如在内存中构建XML数据）。
不要将XML语言狭隘的理解成XML文件。
```

XML语法格式

```

```

1. XML文档声明

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. 标记 (元素 / 标签 / 节点)

XML文档,由一个个的标记组成.

语法:

开始标记(开放标记): <标记名称>

结束标记(闭合标记): </标记名称>

标记名称: 自定义名称，必须遵循以下命名规则:

1.名称可以含字母、数字以及其他的字符

2.名称不能以数字或者标点符号开始

3.名称不能以字符 “xml”（或者 XML、Xml）开始

4.名称不能包含空格，不能包含冒号（:）

5.名称区分大小写

标记内容: 开始标记与结束标记之间 ,是标记的内容.

例如 ,我们通过标记, 描述一个人名:

```
<name>李伟杰</name>
```

3. 一个XML文档中, 必须有且且仅允许有一个根标记.

正例:

```
<names>
    <name>张三</name>
    <name>李四</name>
</names>
```

反例:

```
<name>李四</name>
<name>麻子</name>
```

4. 标记可以嵌套, 但是不允许交叉.

正例:

```
<person>
    <name>李四</name>
    <age>18</age>
</person>
```

反例:

```
<person>
    <name>李四<age></name>
    18</age>
</person>
```

5. 标记的层级称呼 (子标记, 父标记 , 兄弟标记, 后代标记 ,祖先标记)

例如:

```
<persons>
    <person>
        <name>李四</name>
        <length>180cm</length>
    </person>
    <person>
        <name>李四</name>
        <length>200cm</length>
    </person>
</persons>
```

name是person的子标记.也是person的后代标记

name是persons的后代标记.

```
        name是length的兄弟标记.
        person是name的父标记.
        persons是name的祖先标记.

6.  标记名称 允许重复
7.  标记除了开始和结束 , 还有属性.
        标记中的属性, 在标记开始时 描述, 由属性名和属性值 组成.
        格式:
            在开始标记中, 描述属性.
            可以包含0-n个属性, 每一个属性是一个键值对!
            属性名不允许重复 , 键与值之间使用等号连接, 多个属性之间使用空格分割.
            属性值 必须被引号引住.

        案例:
            <persons>
                <person id="10001" groupid="1">
                    <name>李四</name>
                    <age>18</age>
                </person>
                <person id="10002" groupid="1">
                    <name>李四</name>
                    <age>20</age>
                </person>
            </persons>

8.  注释
    注释不能写在文档文档声明前
    注释不能嵌套注释
        格式:
            注释开始:   <!--
            注释结束:   -->
```

案例:

```
描述一组图书books, 至少包含3本书
        图书book包含
            图书名称name
            图书简介info ,
            以及属性id

<?xml version="1.0" encoding="UTF-8"?>
<books>

</books>
```

语法进阶CDATA（了解）

```
CDATA
    CDATA 是不应该由 XML 解析器解析的文本数据。

    像 "<" 和 "&" 字符在 XML 元素中都是非法的。

    "<" 会产生错误，因为解析器会把该字符解释为新元素的开始。

    "&" 会产生错误，因为解析器会把该字符解释为字符实体的开始。
```

某些文本，比如 JavaScript 代码，包含大量 "<" 或 "&" 字符。为了避免错误，可以将脚本代码定义为 CDATA。

CDATA 部分中的所有内容都会被解析器忽略。

CDATA 部分由 "<![CDATA[" 开始，由 "]]>" 结束：

Java解析XML 掌握

面试题 *

问: Java中有几种XML解析方式 ? 分别是什么 ? 有什么样的优缺点 ?

答: 四种.

1. SAX解析

解析方式是事件驱动机制 !

SAX解析器, 逐行读取XML文件解析 , 每当解析到一个标签的开始/结束/内容/属性时,触发事件.

我们可以编写程序在这些事件发生时, 进行相应的处理.

优点:

分析能够立即开始，而不是等待所有的数据被处理

逐行加载,节省内存.有助于解析大于系统内存的文档

有时不必解析整个文档,它可以在某个条件得到满足时停止解析.

缺点:

1. 单向解析，无法定位文档层次，无法同时访问同一文档的不同部分数据(因为逐行解析, 当解析第n行是, 第n-1行已经被释放了, 无法在进行操作了).
2. 无法得知事件发生时元素的层次, 只能自己维护节点的父/子关系.
3. 只读解析方式, 无法修改XML文档的内容.

2. DOM解析

是用与平台和语言无关的方式表示XML文档的官方W3C标准，分析该结构通常需要加载整个文档和内存中建立文档树模型.程序员可以通过操作文档树, 来完成数据的获取 修改 删除等.

优点:

文档在内存中加载, 允许对数据和结构做出更改.

访问是双向的，可以在任何时候在树中双向解析数据。

缺点:

文档全部加载在内存中 , 消耗资源大.

3. JDOM解析

目的是成为Java特定文档模型，它简化与XML的交互并且比使用DOM实现更快。由于是第一个Java特定模型，JDOM一直得到大力推广和促进。

JDOM文档声明其目的是“使用20%（或更少）的精力解决80%（或更多）Java/XML问题”（根据学习曲线假定为20%）

优点:

使用具体类而不是接口，简化了DOM的API。

大量使用了Java集合类，方便了Java开发人员。

缺点:

没有较好的灵活性。
性能不是那么优异。

4. DOM4J解析

它是JDOM的一种智能分支。它合并了许多超出基本XML文档表示的功能，包括集成的XPath支持、XML Schema支持以及用于大文档或流化文档的基于事件的处理。它还提供了构建文档表示的选项，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。如今你可以看到越来越多的Java软件都在使用DOM4J来读写XML。

目前许多开源项目中大量采用DOM4J ，例如:Hibernate

DOM4J解析XML 掌握

步骤:

1. 引入jar文件 dom4j.jar
2. 创建一个指向XML文件的输入流

```
FileInputStream fis = new FileInputStream("xml文件的地址");
```

3. 创建一个XML读取工具对象

```
SAXReader sr = new SAXReader();
```

4. 使用读取工具对象, 读取XML文档的输入流 , 并得到文档对象

```
Document doc = sr.read(fis);
```

5. 通过文档对象, 获取XML文档中的根元素对象

```
Element root = doc.getRootElement();
```

文档对象 Document

指的是加载到内存的 整个XML文档.

常用方法:

1. 通过文档对象, 获取XML文档中的根元素对象

```
Element root = doc.getRootElement();
```

2. 添加根节点

```
Element root = doc.addElement("根节点名称");
```

元素对象 Element

指的是XML文档中的单个节点.

常用方法:

1. 获取节点名称

```
String getName();
```

2. 获取节点内容

```
String getText();
```

3. 设置节点内容

```
String setText();
```

4. 根据子节点的名称 , 获取匹配名称的第一个子节点对象.

```
Element element(String 子节点名称);
```

5. 获取所有的子节点对象

```
List<Element> elements();
```

6. 获取节点的属性值

```
String attributeValue(String 属性名称);
```

7. 获取子节点的内容

```
String elementText(String 子节点名称);
```

8. 添加子节点

```
Element addElement(String 子节点名称);
```

```
    9.  添加属性
            void addAttribute(String 属性名,String 属性值);
```

解析本地文件案例:

```
    //1.    获取文件的输入流
    FileInputStream fis = new
FileInputStream("C:\\code\\35\\code1\\day03_XML\\src\\books.xml");
    //2.    创建XML读取工具对象
    SAXReader sr = new SAXReader();
    //3.    通过读取工具, 读取XML文档的输入流 , 并得到文档对象
    Document doc = sr.read(fis);
    //4.    通过文档对象 , 获取文档的根节点对象
    Element root = doc.getRootElement();
    //5.    通过根节点, 获取所有子节点
    List<Element> es = root.elements();
    //6.    循环遍历三个book
    for (Element e : es) {
        //1.    获取id属性值
        String id = e.attributeValue("id");
        //2.    获取子节点name , 并获取它的内容
        String name = e.element("name").getText();
        //3.    获取子节点info , 并获取它的内容
        String info = e.element("info").getText();
        System.out.println("id="+id+",name="+name+",info="+info);
    }
```

解析网络文件案例:

```
        String phone = "18516955565";
        //1.    获取到XML资源的输入流
        URL url = new URL("http://apis.juhe.cn/mobile/get?
phone="+phone+"&dtype=xml&key=9f3923e8f87f1ea50ed4ec8c39cc9253");
        URLConnection conn = url.openConnection();
        InputStream is = conn.getInputStream();
        //2.    创建一个XML读取对象
        SAXReader sr = new SAXReader();
        //3.    通过读取对象 读取XML数据, 并返回文档对象
        Document doc = sr.read(is);
        //4.    获取根节点
        Element root = doc.getRootElement();
        //5.    解析内容
        String code = root.elementText("resultcode");
        if("200".equals(code)){
            Element result = root.element("result");
            String province = result.elementText("province");
            String city = result.elementText("city");
            if(province.equals(city)){
                System.out.println("手机号码归属地为: "+city);
            }else{
                System.out.println("手机号码归属地为: "+province+" "+city);
            }
        }else{
            System.out.println("请输入正确的手机号码");
        }
```

DOM4J - XPATH解析XML

路径表达式

```
通过路径快速的查找一个或一组元素

路径表达式:
    1.  /   :   从根节点开始查找
    2.  //  :   从发起查找的节点位置 查找后代节点  ***
    3.  .   :   查找当前节点
    4.  ..  :   查找父节点
    5.  @   :   选择属性. *
                属性使用方式:
                [@属性名='值']
                [@属性名>'值']
                [@属性名<'值']
                [@属性名!='值']

    books:   路径:   //book[@id='1']//name
    books
        book id=1
            name
            info
        book id=2
            name
            info
```

使用步骤

```
通过Node类的两个方法, 来完成查找:
(Node是 Document 与 Element 的父接口)

方法1.
    //根据路径表达式, 查找匹配的单个节点
    Element e = selectSingleNode("路径表达式");

方法2.
    List<Element> es = selectNodes("路径表达式");


案例:

        String phone = "18313935565";
        //1.    获取到XML资源的输入流
        URL url = new URL("http://apis.juhe.cn/mobile/get?
phone="+phone+"&dtype=xml&key=9f3923e8f87f1ea50ed4ec8c39cc9253");
        URLConnection conn = url.openConnection();
        InputStream is = conn.getInputStream();
        //2.    创建一个XML读取对象
        SAXReader sr = new SAXReader();
        //3.    通过读取对象 读取XML数据, 并返回文档对象
        Document doc = sr.read(is);

        Node node = doc.selectSingleNode("//company");
        System.out.println("运营商: "+node.getText());
        is.close();
```

Java生成XML 熟悉

步骤:

1. 通过文档帮助器 (DocumentHelper) , 创建空的文档对象
   ```
   Document doc = DocumentHelper.createDocument();
   ```
2. 通过文档对象, 向其中添加根节点
   ```
   Element root = doc.addElement("根节点名称");
   ```
3. 通过根节点对象root , 丰富我们的子节点
   ```
   Element e = root.addElement("元素名称");
   ```
4. 创建一个文件输出流 ,用于存储XML文件
   ```
   FileOutputStream fos = new FileOutputStream("要存储的位置");
   ```
5. 将文件输出流, 转换为XML文档输出流
   ```
   XMLWriter xw = new XMLWriter(fos);
   ```
6. 写出文档
   ```
   xw.write(doc);
   ```
7. 释放资源
   ```
   xw.close();
   ```

案例:

```
        //1.    通过文档帮助器, 创建空的文档对象
        Document doc = DocumentHelper.createDocument();
        //2.    向文档对象中, 加入根节点对象
        Element books = doc.addElement("books");
        //3.    向根节点中 丰富子节点
        for(int i=0;i<1000;i++) {
            //向根节点中加入1000个book节点.
            Element book = books.addElement("book");
            //向book节点, 加入id属性
            book.addAttribute("id", 1+i+"");
            //向book节点中加入name和info节点
            Element name = book.addElement("name");
            Element info = book.addElement("info");
            name.setText("苹果"+i);
            info.setText("哈哈哈"+i);
        }
        //4.    创建文件的输出流
        FileOutputStream fos = new FileOutputStream("c:\\books.xml");
        //5.    将文件输出流 , 转换为XML文档输出流
        XMLWriter xw = new XMLWriter(fos);
        //6.    写出XML文档
        xw.write(doc);
        //7.    释放资源
        xw.close();
        System.out.println("代码执行完毕");
```

XStream 的使用 了解

快速的将Java中的对象, 转换为 XML字符串.

使用步骤:

```
    1.  创建XStream 对象
        XStream x = new XStream();

    [2].修改类生成的节点名称 (默认节点名称为 包名.类名)
        x.alias("节点名称",类名.class);

    3.  传入对象 , 生成XML字符串
        String xml字符串 = x.toXML(对象);

案例:
    Person p = new Person(1001, "张三", "不详");
    XStream x = new XStream();
    x.alias("haha", Person.class);
    String xml = x.toXML(p);
    System.out.println(xml);
```

JSON

```
简介:
    JSON:   JavaScript Object Notation  JS对象简谱 , 是一种轻量级的数据交换格式.
```

对象格式

```
    一本书
        书名
        简介
    java
        class Book{
            private String name;
            private String info;
```

```
            get/set...
        }
        Book b = new Book();
        b.setName(“金苹果”);
        b.setInfo(“种苹果”);
        ...

    js:
        var b = new Object();
        b.name = "金苹果";
        b.info = "种苹果";

    XML:
        <book>
            <name>金苹果</name>
            <info>种苹果</info>
        </book>
    JSON:
        {
            "name":"金苹果",
            "info":"种苹果"
        }


    一个对象，由一个大括号表示.
        括号中 描述对象的属性 .  通过键值对来描述对象的属性
        (可以理解为，大括号中，包含的是一个个的键值对.)
        格式:
            键与值之间使用冒号连接，多个键值对之间使用逗号分隔.
            键值对的键 应使用引号引住 (通常Java解析时，键不使用引号会报错. 而JS能正确解析.)
            键值对的值，可以是JS中的任意类型的数据
```

数组格式

```
在JSON格式中可以与对象互相嵌套
[元素1,元素2...]
```

案例

```
            {
                "name":"伟杰老师",
                "age":18,
                "pengyou":["张三","李四","王二","麻子",{
                            "name":"野马老师",
                            "info":"像匹野马一样狂奔在技术钻研的道路上"
                        }],
                "heihei":{
                    "name":"大长刀",
                    "length":"40m"
                }
            }
```

Java与JSON

```
做什么?
    将Java中的对象 快速的转换为 JSON格式的字符串.
    将JSON格式的字符串, 转换为Java的对象.
```

Gson

- 将对象转换为JSON字符串

```
转换JSON字符串的步骤:
    1.  引入JAR包
    2.  在需要转换JSON字符串的位置编写如下代码即可:
        String json = new Gson().toJSON(要转换的对象);

案例:
    Book b = BookDao.find();
    String json = new Gson().toJson(b);
    System.out.println(json);
```

- 将JSON字符串转换为对象

```
    1.  引入JAR包
    2.  在需要转换Java对象的位置, 编写如下代码:
        对象 = new Gson().fromJson(JSON字符串,对象类型.class);

案例:
    String json = "{\"id\":1,\"name\":\"金苹果\",\"author\":\"李伟杰
\",\"info\":\"嘿嘿嘿嘿嘿嘿\",\"price\":198.0}";
    Book book = new Gson().fromJson(json, Book.class);
    System.out.println(book);
```

FastJson

- 将对象转换为JSON字符串

```
转换JSON字符串的步骤:
    1.  引入JAR包
    2.  在需要转换JSON字符串的位置编写如下代码即可:
        String json=JSON.toJSONString(要转换的对象);

案例:
    Book b = BookDao.find();
    String json=JSON.toJSONString(b);
    System.out.println(json);
```

- 将JSON字符串转换为对象

1. 引入JAR包
2. 在需要转换Java对象的位置，编写如下代码：

```
类型 对象名=JSON.parseObject(JSON字符串，类型.class);
    或
List<类型> list=JSON.parseArray(JSON字符串,类型.class);
```

案例：

```
String json = "{\"id\":1,\"name\":\"金苹果\",\"author\":\"李伟杰
\",\"info\":\"嘿嘿嘿嘿嘿嘿\",\"price\":198.0}";
Book book = JSON.parseObject(json, Book.class);
System.out.println(book);
```

####