第六章 XML

课前回顾

1. 网络通信有哪些部分组成

```
协议： TCP/UDP
IP：定位计算机
端口：定位应用
URL：统一资源定位符
```

2. TCP和UDP有什么区别

```
TCP是一个点对点的通信端点，属于可靠性信息传输。UDP是无连接的不可靠信息传输。
```

章节内容

- XML文档 **重点**
- XML文档约束 **熟悉**
- XML解析 **重点**

章节目标

- 掌握XML的规范书写
- 掌握XML中的特殊字符处理
- 了解XML约束
- 掌握XML解析

第一节 XML

1. 什么是XML

```
XML全称是Extensible Markup Language,可扩展标记语言
```

2. XML语法

a. xml是一种文本文档，其后缀名为xml

b. xml文档内容的第一行必须是对整个文档类型的声明

c.xml文档内容中有且仅有一个根标签

d.xml文档内容中的标签必须严格闭合

e. xml文档内容中的标签属性值必须使用单引号或者双引号引起来

f. xml文档内容中的标签名区分大小写

示例

```xml
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student name="张三" age="20" sex="男"></student>
    <STUDENT>
        <NAME>张三</NAME>
        <AGE>20</AGE>
        <SEX>男</SEX>
    </STUDENT>
</students>
```

思考：如果在XML文档内容中出现了像(<)这类似的特殊符号，怎么办?

使用标签CDATA来完成，CDATA标签中的内容会按原样展示

语法

```xml
<![CDATA[
    <!--内容 -->
]]>
```

XML文档可以自定义标签，为了更规范的使用XML文档，可以使用XML约束来限定XML文档中的标签使用。

XML约束可以通过DTD文档和Schema文档来实现。其中DTD文档比较简单，后缀名为dtd，而Schema技术则比较复杂，后缀名为xsd。

第二节 DTD约束

1. DTD约束元素

元素类型

EMPTY（空元素），元素不包含任何数据，但是可以有属性，如:

```xml
<student name="李四" sex="女" age="21" />
```

#PCDATA（字符串），PCDATA是指被解析器解析的文本也就是字符串内容，不能包含其他类型的元素，如:

```xml
<name>张三</name>
<sex>男</sex>
<age>20</age>
```

ANY（任何内容都可以）

DTD约束元素出现顺序及次数

情景	语法	描述
顺序出现	`<!ELEMENT name (a, b)>`	子元素a、b必须同时出现，且a必须在b之前出现
选择出现	`<!ELEMENT name (a|b)>`	子元素a、b只能有一个出现，要么是a，要么是b
只出现一次	`<!ELEMENT name (a)>`	子元素a只能且必须出现一次
一次或多次	`<!ELEMENT name (a+)>`	子元素a要么出现一次，要么出现多次
零次或多次	`<!ELEMENT name (a*)>`	子元素a可以出现任意次（包括不出现，即出现零次）
零次或一次	`<!ELEMENT name (a?)>`	子元素a可以出现一次或不出

元素格式

```
<!ELEMENT 元素名称 元素类型>
```

示例

```
<!ELEMENT students (student*)>
<!ELEMENT student (name,sex,age)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

2. DTD约束元素属性

属性值类型

CDATA，属性值为普通文本字符串。

Enumerated，属性值的类型是一组取值的列表，XML文件中设置的属性值只能是这个列表中的某一个值。

ID，表示属性值必须唯一，且不能以数字开头

属性值设置

#REQUIRED，必须设置该属性。

#IMPLIED，该属性可以设置也可以不设置。

#FIXED，该属性的值为固定的。

使用默认值。

属性格式

```
<!ATTLIST 元素名 属性名 属性值类型 设置说明>
```

示例

```
<!ELEMENT students (student*)>
<!ELEMENT student (name,sex,age)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ATTLIST student number ID #REQUIRED>
<!ATTLIST student name CDATA>
<!ATTLIST student sex(男|女|其他) #IMPLIED>
<!ATTLIST student age CDATA>
```

3. 引入DTD约束

语法

```
<!DOCTYPE 根标签名 SYSTEM "约束文档名.dtd">
```

示例

```
<!ELEMENT students (student*)>
<!--<!ELEMENT student (name, sex, age) ANY>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT age (#PCDATA)>-->
<!ELEMENT student  EMPTY>
<!ATTLIST student number ID #REQUIRED>
<!ATTLIST student name CDATA>
<!ATTLIST student sex(男|女|其他) #IMPLIED>
<!ATTLIST student age CDATA>
<!ATTLIST student country(中国) CDATA #FIXED>
```

第三节 XML解析

1. 解析方式

DOM解析

DOM解析将XML文档一次性加载进内存，在内存中形成一颗dom树，优点是操作方便，可以对文档进行CRUD的所有操作，缺点就是占内存

SAX解析

SAX解析式将XML文档逐行读取，是基于事件驱动的，优点是不占内存，缺点是只能读取，不能增删改

对于XML操作，我们通常都是读取操作，增删改的情况较少，因此这里主要讲解SAX解析,SAX解析最优实现是Dom4j解析器。

2. `Dom4j` 解析XML

示例

```
package com.cyx.xml;

import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.DocumentException;
```

```
import org.dom4j.Element;
import org.dom4j.io.SAXReader;

import java.io.InputStream;
import java.util.Iterator;

public class XmlParser {

    public static void main(String[] args) {
        //构建一个SAX读取器
        SAXReader reader = new SAXReader();
        //通过类的字节码对象获取一个给定资源并将该资源读取到流的通道中
        InputStream is =
XmlParser.class.getResourceAsStream("student.xml");
        try {
            //SAX读取器从通道中读取一个文档对象
            Document document = reader.read(is);
            //获取文档的根元素，因为XML文档只会有一个根元素
            Element root = document.getRootElement();
            //获取根元素的标签名
            String tagName = root.getQualifiedName();
            System.out.println("XML文档根标签：" + tagName);
            //获取根元素的下一级子元素
//            List<Element> elements = root.elements();
//            for(Element element: elements){
//                //获取元素的标签名
//                String tag = element.getQualifiedName();
//                System.out.println(tag);
//                //获取元素的所有属性
//                List<Attribute> attributes = element.attributes();
//                for(Attribute attr: attributes){
//                    //获取属性名
//                    String name = attr.getName();
//                    //获取属性值
//                    String value = attr.getValue();
//                    System.out.print("属性：" + name + "=>" + value +
"\t");
//                }
//                System.out.println();
//            }
            Iterator<Element> iterator = root.elementIterator("student");
            while (iterator.hasNext()){
                Element element = iterator.next();
                //获取元素的标签名
                String tag = element.getQualifiedName();
                System.out.println(tag);
//                //获取元素的所有属性
//                List<Attribute> attributes = element.attributes();
//                for(Attribute attr: attributes){
//                    //获取属性名
//                    String name = attr.getName();
//                    //获取属性值
//                    String value = attr.getValue();
//                    System.out.print("属性：" + name + "=>" + value +
"\t");
//                }
//                System.out.println();
                String name = element.attributeValue("name");
```

```
                String sex = element.attributeValue("sex");
                String age = element.attributeValue("age");
                System.out.println(name + "\t" + sex + "\t" + age);
            }
        } catch (DocumentException e) {
            e.printStackTrace();
        }
    }
}
```