day26-xml和动态代理

一,XML入门【熟悉】

1.xml概述

- 可扩展标记语言, 标签可以自定义的

2.xml和html区别

- html所有标签都是预定义的，xml所有标签都是自定义的
- html语法松散，xml语法严格,区分大小写
- html做页面展示，xml描述数据

3.xml作用

- 作为配置文件。
- 存储数据。
- 用来传输数据.

4.xml语法规范

4.1文档声明

```
<?xml version = "1.0" encoding = "utf-8" standalone="yes" ?>
```

- 必须写在xml文档的第一行第一列
- 写法：<?xml version="1.0" ?>
- 属性:
    1. version：版本号 固定值 1.0
    2. encoding:指定文档的码表。默认值为 iso-8859-1
    3. standalone：指定文档是否独立 yes 或 no(了解)

4.2元素：xml文档中的标签

- 文档中必须有且只能有一个根元素,其他标签都是这个根标签的子标签或孙标签
- 元素需要正确闭合
    1. 包含标签主体：somecontent
    2. 不含标签主体：
- 元素需要正确嵌套,不允许有交叉嵌套。
- 元素名称要遵守
    1. 元素名称区分大小写
    2. 数字不能开头

3. 标签之间不能有空格,特殊字符

4.3属性

- 属性值必须用引号引起来。单双引号都行
- 在一个标签里面属性不能重复

4.4注释

语法

快捷键: Ctrl+Shift+/

- 文档声明前面不能有注释
- 注释不能嵌套

4.5特殊字符和CDATA区

1.在 XML 中有 5 个预定义的实体引用：

&lt;	<	小于
&gt;	>	大于
&amp;	&	和号
&apos;	'	省略号
&quot;	"	引号

2.CDATA 内部的所有东西都会被解析器忽略,当做文本

语法:<![CDATA[内容]]>

二,xml约束【了解】

1.xml约束概述

- 约束就是xml的书写规则,
- 约束文档定义了在XML中允许出现的元素(标签)名称、属性及元素(标签)出现的顺序等等。

2.常用约束分类

2.1dtd约束

2.1.1 引入方式

1. 内部引入

```
<!DOCTYPE 根元素 [元素声明]>
```

2. 外部引入; dtd是一个独立的文件 .dtd
    - 本地DTD

```
<!DOCTYPE 根元素 SYSTEM "文件名">
```

- 网络DTD

```
<!DOCTYPE 根元素 PUBLIC "DTD名称" "DTD文档的URL">
```

2.1.2语法

1. 元素：<!ELEMENT 元素名称 元素组成>

- 元素组成：EMPTY，ANY，(子元素)，(#PCDATA)
- 如果出现子元素: ?(出现0次或1次),
  *(出现0次或多次),
  +(出现1次或多次) 表示子元素出现的个数.
  |或者（只能出现一个）
  , 代表子元素出现必须按照顺序

2. 属性<!ATTLIST 元素名称 属性名称 属性类型 属性使用规则>

- 属性类型: ID,枚举,CDATA
- 使用规则: #REQUIRED:属性值是必需的
  #IMPLIED :属性不是必需的
  #FIXED value:属性值是固定的

ID使用：字母开头

2.2schema

2.2.1 dtd和**schema**的区别

- XMLSchema符合XML语法结构。
- DOM、SAX等XMLAPI很容易解析出XML Schema文档中的内容。
- XMLSchema对名称空间支持得非常好。
- XMLSchema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。
- XMLSchema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。
- 但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。
- XMLSchema不能像DTD一样定义实体，比DTD更复杂，

2.2.1根据**schema**约束写**xml**文件步骤

- 查出Schema文档,找出根元素
- 根元素来自哪个名称空间
- 这个名称空间和哪个xsd文件对应(指定约束的路径)
- schemaLocation不是关键字，来自一个标准的名称空间

三,XML解析【掌握】

1.xml解析方式

1.1DOM解析

将xml文档加载到内存，形成一颗dom树(document对象)，将文档的各个组成部封装为一些对象。

优点: 因为，在内存中会形成dom树，可以对dom树进行增删改查。

缺点: 如果xml文件太大), dom树非常占内存，解析速度慢。

Document:文档节点

Element:元素(标签)节点

Text:文本节点

Attribute:属性节点

Comment:注释节点

1.2SAX解析

逐行读取，基于事件(函数或者方法)驱动

优点：内存占用很小，速度快

缺点：只能读取，不能回写

2.常用xml解析器(jar包)

JAXP：sun公司提供的解析。支持dom和sax。

JDOM

DOM4J：document for java 民间方式，但是是事实方式。非常好。 支持dom和sax.

JSOUP: 爬虫, 解析html

3.Dom4J的使用【重点】

3.1使用步骤

1. 导入jar包 dom4j-1.6.1j.jar
2. 创建解析器

   SAXReader reader = new SAXReader();
3. 读取xml 获得document对象

   Document document = reader.read(xml对应的流)
4. 得到根元素

   Element rootElement = document.getRootElement();

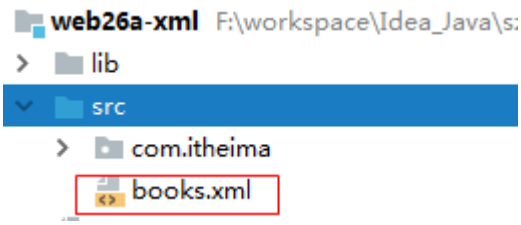
3.2常见的API

- List list=element.elements(); 获取所有子元素节点

- element.attributeValue("属性名")或者element.attribute("属性名").getValue(); 获取节点属性的值
- element.getText(); 获得节点的文本值

3.3示例代码

- xml(把xml放在src目录下, 方便读取)




```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <book id="b1">
        <name>四十二章经</name>
        <autor>不详</autor>
        <price>10000</price>
    </book>
    <book id="b2">
        <name>葵花宝典</name>
        <autor>小李子</autor>
        <price>8888</price>
    </book>
    <apple>
        <name>苹果</name>
    </apple>
</books>
```

- Dom4JDemo

```java
public class Dom4JDemo {

    @Test
    //步骤
    public void fun01() throws Exception {
        //1. 创建SaxReader对象
        SAXReader saxReader = new SAXReader();
        //2. 读取需要解析的xml文件, 获得文档对象
        InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
        Document document = saxReader.read(is);
        //3. 获得根元素 books
        Element rootElement = document.getRootElement();


    }

    // 获得第一本书id的属性值 b1
    @Test
    public void fun02() throws Exception {
        //1. 创建SaxReader对象
        SAXReader saxReader = new SAXReader();
        //2. 读取需要解析的xml文件, 获得文档对象
        InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
        Document document = saxReader.read(is);
        //3. 获得根元素 books
        Element rootElement = document.getRootElement();
        //4. 获得根元素(books)所有的 子标签
        List<Element> elements = rootElement.elements();
        //5. 获得第一本book(第一个孩子)
        Element bookEle = elements.get(0);
        //6.获得获得第一本书id的属性值
        String id = bookEle.attributeValue("id");
        System.out.println(id);

        is.close();


    }

    //获得第二本书 书的名字 葵花宝典

    @Test
    public void fun03() throws Exception {
        //1. 创建SaxReader对象
        SAXReader saxReader = new SAXReader();
        //2. 读取需要解析的xml文件, 获得文档对象
        InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
        Document document = saxReader.read(is);
        //3. 获得根元素 books
        Element rootElement = document.getRootElement();

        //4.获得第二本数

       Element bookEle = (Element) rootElement.elements().get(1);
```

```
        //5. 获得第二本书 标签里面的name标签(第一个孩子)
         Element nameEle = (Element) bookEle.elements().get(0);

         //6. 获得name标签里面的文本
         System.out.println(nameEle.getText());

         is.close();
    }


}
```

4.XPath的使用,基于dom4J

4.1介绍

Xpath定义了一种规则,专门用于查询xml, 可以认为xpath是dom4J的扩展

4.2使用步骤

1. 导入jar包 jaxen...jar(当前资料里是: jaxen-1.1-beta-6.jar) 注意: 两个(dom4j的jar包和xpath的jar包)
2. 创建解析器

   SAXReader reader = new SAXReader();
3. 解析xml 获得document对象

   Document document = reader.read(is);

4.3常见的API

- document.selectSingleNode("xpath语法"); 获得一个节点(标签,元素)
- document.selectNodes("xpath语法"); 获得多个节点(标签,元素)

4.4示例代码

```
public class XpathDemo {

    @Test
    //环境准备
    public void fun01() throws Exception{
        //1. 创建解析器对象
        SAXReader saxReader = new SAXReader();
        //2. 读取要解析的xml文件, 得到document
       InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
      Document document = saxReader.read(is);
        //3. 根据xpath, 获得标签
        //获得一个
        document.selectSingleNode("xpath语法");
       //获得多个
        document.selectNodes("xpath语法");
    }

    @Test
    //获得apple标签里面name标签的值  苹果
    public void fun02() throws Exception{
        //1. 创建解析器对象
        SAXReader saxReader = new SAXReader();
        //2. 读取要解析的xml文件, 得到document
        InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
       Document document = saxReader.read(is);
        //3. 根据xpath, 获得标签
        Element nameEle  = (Element) document.selectSingleNode("/books/apple/name");
        //4. 获得值
        System.out.println(nameEle.getText());

    }

    @Test
    //获得apple标签里面name标签的值  苹果
    public void fun03() throws Exception{
        //1. 创建解析器对象
        SAXReader saxReader = new SAXReader();
        //2. 读取要解析的xml文件, 得到document
        InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
       Document document = saxReader.read(is);
        //3. 根据xpath, 获得标签
        Element nameEle  = (Element) document.selectSingleNode("//apple/name");
        //4. 获得值
        System.out.println(nameEle.getText());

    }

    @Test
    //获得第一本书的 id属性值 b1
    public void fun04() throws Exception{
        Class clazz = Class.forName("");


        //1. 创建解析器对象
```

```
            SAXReader saxReader = new SAXReader();
            //2. 读取要解析的xml文件, 得到document
            InputStream is = Dom4JDemo.class.getClassLoader().getResourceAsStream("books.xml");
           Document document = saxReader.read(is);
            //3. 根据xpath, 获得标签
            Element booEle  = (Element) document.selectSingleNode("//book[1]");
            //4. 获得id的属性值
            System.out.println(booEle.attributeValue("id"));


        }

    }
```

四,动态代理

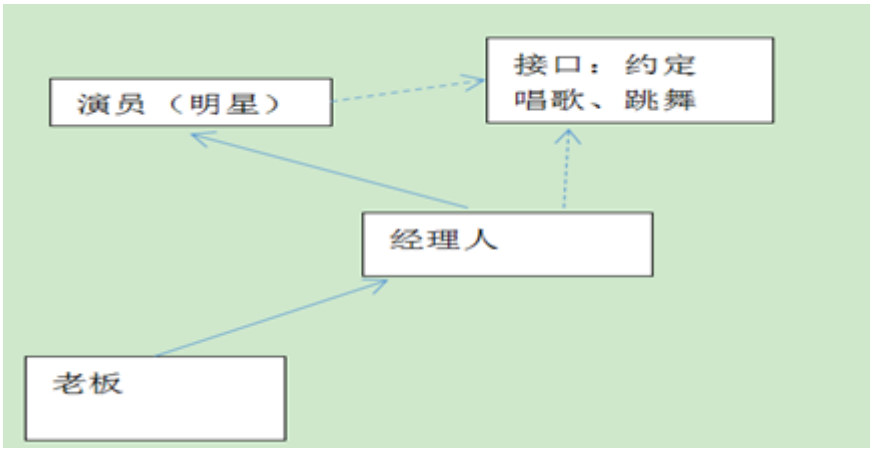
1.代理模式概述

1.1什么是代理模式

ProxyPattern（即：代理模式），23种常用的面向对象软件的设计模式之一

代理模式的定义：为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用。

作用：增强一个类中的某个方法.对程序进行扩展. Spring框架中AOP.



1.2动态代理介绍

动态代理它可以直接给某一个目标(被代理 对象)对象(实现了某个或者某些接口)生成一个代理对象，而不需要代理类存在。

动态代理与代理模式原理是一样的，只是它没有具体的代理类，直接通过反射生成了一个代理对象。

- 动态代理的分类

  jdk提供一个Proxy类可以直接给实现接口类的对象直接生成代理对象

  spring中动态代理:cglib

2.jdk中的动态代理的使用

2.1API介绍

Java.lang.reflect.Proxy类可以直接生成一个代理对象

- Proxy.newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)生成一个代理对象
    - 参数1:ClassLoader loader 代理对象的类加载器 一般使用被代理对象的类加载器
    - 参数2:Class<?>[] interfaces 代理对象的要实现的接口 一般使用的被代理对象实现的接口
    - 参数3:InvocationHandler h (接口)执行处理类
- InvocationHandler中的invoke(Object proxy, Method method, Object[] args)方法：调用代理类的任何方法，此方法都会执行
    - 参数3.1:代理对象(慎用)
    - 参数3.2:当前执行的方法
    - 参数3.3:当前执行的方法运行时传递过来的参数
        - 返回值:当前方法执行的返回值

2.2代码实现

```
package com.itheima.proxy.dynamic;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

import org.junit.Test;

public class Demo01 {

    @Test
    public void fun01(){
        final Qq qq = new Qq();

        Car car = (Car) Proxy.newProxyInstance(qq.getClass().getClassLoader(),
qq.getClass().getInterfaces(), new InvocationHandler() {

            @Override
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {



                if("run".equals(method.getName())){
                    System.out.println("QQ加了鸡血可以跑100迈...");
                    return null;
                }

                Object result = method.invoke(qq, args);
                System.out.println("result="+result);
                return "哈哈哈";
            }
        });

        //car.run();
        String str = car.addOil(93);
        System.out.println("str="+str);
    }


    interface Car{
        void run();
        void stop();

        String  addOil(int num);
    }

    class Qq implements Car{

        @Override
        public void run() {
            System.out.println("Qq可以跑60迈...");
        }
```

```
        @Override
        public void stop() {
            System.out.println("Qq刹车....");
        }

        @Override
        public String addOil(int num) {
            return "qq加"+num+"号油...";

        }

    }
}
```