

任务七_XML

1. XML基本介绍

1.1 概述

XML即可扩展标记语言 (Extensible Markup Language)

- W3C在1998年2月发布1.0版本，2004年2月又发布1.1版本，但因为1.1版本不能向下兼容1.0版本，所以1.1没有人用。同时，在2004年2月W3C又发布了1.0版本的第三版。我们要学习的还是1.0版本！

特点

- 可扩展的, 标签都是自定义的
- 语法十分严格

1.2 XML的作用

XML能做什么？

功能	说明
存储数据	通常，我们在数据库中存储数据。不过，如果希望数据的可移植性更强，我们可以把数据存储 XML 文件中
配置文件	作为各种技术框架的配置文件使用 (最多)
在网络中传输	客户端可以使用XML格式向服务器端发送数据,服务器接收到xml格式数据,进行解析

2. XML的语法

2.1 XML文档声明格式

- 文档声明必须为结束；
- 文档声明必写在第一行；

1) 语法格式:

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

2) 属性说明:

- **version**: 指定XML文档版本。必须属性，因为我们不会选择1.1，只会选择1.0；
- **encoding**: 指定当前文档的编码。可选属性，默认值是utf-8；

2.2 元素

Element 元素: 是XML文档中最重要的组成部分

元素的命名规则

1. 不能使用空格，不能使用冒号
2. xml 标签名称区分大小写
3. XML 必须有且只有一个根元素

语法格式:

```
<users><users>
```

1) XML 必须有且只有一个根元素，它是所有其他元素的父元素，比如以下实例中 users 就是根元素：

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

</users>
```

2) 普通元素的结构开始标签、元素体、结束标签组成。

```
<hello> 大家好 </hello>
```

3) 元素体：元素体可以是元素，也可以是文本

```
<hello>
    <a>你好</a>
</hello>
```

4) 空元素：空元素只有开始标签，而没有结束标签，但元素必须自己闭合

```
<close/>
```

2.3 属性

```
<bean id="" class=""> </bean>
```

1. 属性是元素的一部分，它必须出现在元素的开始标签中
2. 属性的定义格式：属性名=属性值，其中属性值必须使用单引或双引
3. 一个元素可以有0~N个属性，但一个元素中不能出现同名属性
4. 属性名不能使用空格、冒号等特殊字符，且必须以字母开头

2.4 注释

XML的注释，以“<!--”开始，以“-->”结束。注释内容会被XML解析器忽略！

2.5 使用XML 描述数据表中的数据

<input type="checkbox"/>	eid	ename	age	sex	salary	empdate
<input type="checkbox"/>	2	林黛玉	20	女	5000	2019-03-14
<input type="checkbox"/>	3	杜甫	40	男	6000	2020-01-01
<input type="checkbox"/>	4	李白	25	男	3000	2017-10-01
<input type="checkbox"/>	5	张百万	20	女	15000	1990-12-26

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

```

<employees>

    <employee eid="2">
        <ename>林黛玉</ename>
        <age>20</age>
        <sex>女</sex>
        <salary>5000</salary>
        <empdate>2019-03-14</empdate>
    </employee>

    <employee eid="3">
        <ename>杜甫</ename>
        <age>40</age>
        <sex>男</sex>
        <salary>15000</salary>
        <empdate>2010-01-01</empdate>
    </employee>

</employees>

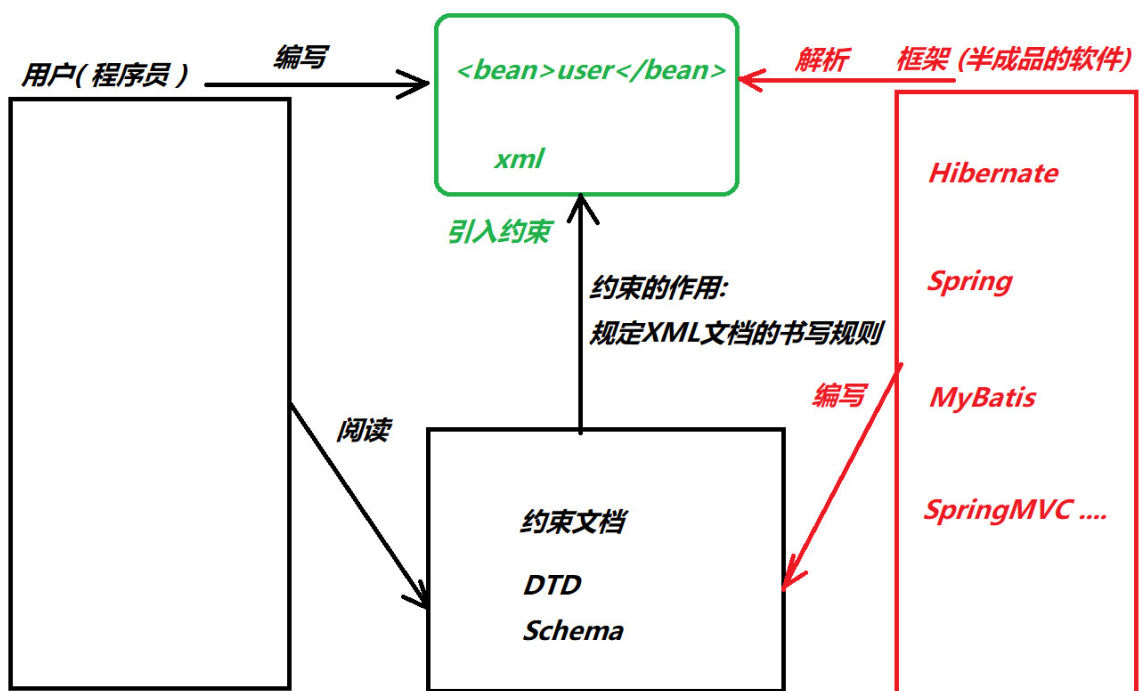
```

3. XML约束

- 在XML技术里，可以编写一个文档来约束一个XML文档的书写规范，这称之为XML约束。
- 常见的xml约束：
 - DTD
 - Schema
- 作为程序员只要掌握两点
 - 会阅读
 - 会引入
 - 不用自己编写

XML由谁来编写？ -- 用户,指的就是客户程序员

XML由谁来解析？ -- 软件 (各种框架,都可以解析XML)



3.1 DTD约束

DTD (Document Type Definition) , 文档类型定义, 用来约束XML文档。规定XML文档中元素的名称, 子元素的名称及顺序, 元素的属性等。

3.1.1 编写DTD

- 开发中, 我们不会自己编写DTD约束文档
- 常情况我们都是通过框架提供的DTD约束文档, 编写对应的XML文档。常见框架使用DTD约束有: Struts2、hibernate等。

创建约束文件 student.dtd

```
<!--
ELEMENT: 用来定义元素

students (student+) : 代表根元素 必须是 <students>

student+ : 根标签中至少有一个 student子元素, + 代表至少一个

student (name,age,sex): student 标签中包含的子元素,按顺序出现

#PCDATA: 是普通文本内容

ATTLIST: 用来定义属性
student number ID #REQUIRED
student子元素中 有一个ID属性叫做 number,是必须填写的
ID: 唯一 值只能是字母或者下划线开头
-->
```

3.1.2 引入DTD

- 引入dtd文档到xml文档中,两种方式
 - 内部dtd: 将约束规则定义在xml文档中
 - 外部dtd: 将约束的规则定义在外部的dtd文件中

本地:
网络:

- student.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE students SYSTEM "student.dtd">
<students>

  <student number="s1">
    <name>小斌</name>
    <age>22</age>
    <sex>男</sex>
```

```

</student>

<student number="s2">
  <name>广坤</name>
  <age>55</age>
  <sex>男</sex>
</student>

</students>

```

3.2 Schema约束

3.2.1 什么是Schema

1. Schema是新的XML文档约束, 比DTD强大很多, 是DTD 替代者;
2. Schema本身也是XML文档, 但Schema文档的扩展名为xsd, 而不是xml。
3. Schema 功能更强大, 内置多种简单和复杂的数据类型
4. Schema 支持命名空间 (一个XML中可以引入多个约束文档)

3.2.2 Schema约束示例

student.xsd

```

<?xml version="1.0"?>
<xsd:schema xmlns="http://www.lagou.com/xml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.lagou.com/xml"
  elementFormDefault="qualified">

  <xsd:element name="students" type="studentsType"/>
  <xsd:complexType name="studentsType">
    <xsd:sequence>
      <xsd:element name="student" type="studentType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="studentType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="age" type="ageType" />
      <xsd:element name="sex" type="sexType" />
    </xsd:sequence>
    <xsd:attribute name="number" type="numberType" use="required"/>
  </xsd:complexType>
  <xsd:simpleType name="sexType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="male"/>
      <xsd:enumeration value="female"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ageType">
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="200"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="numberType">

```

```

        <xsd:restriction base="xsd:string">
            <xsd:pattern value="hehe_\d{4}"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>

```

Xml Schema的根元素:

```

<?xml version="1.0"?>
<xsd:schema xmlns="http://www.lagou.com/xml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.lagou.com/xml" elementFormDefault="qualified">

```

① 表示此文档默认的命名空间是什么

② 表示数据类型等定义 来自w3

③ 表示文档中要定义的元素,来自哪个命名空间

④ 表示要求xml文档的每一个元素都要有命名空间指定

3.2.3 XML引入Schema约束

xml中引入schema约束的步骤:

1) 查看schema文档, 找到根元素, 在xml中写出来

```

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

</students>

```

2) 根元素来自哪个命名空间。使用xmlns指令来声明

```

<?xml version="1.0" encoding="UTF-8" ?>
<students
  xmlns="http://www.lagou.com/xml"
>

</students>

```

3) 引入 w3c的标准命名空间, 复制即可

```

<?xml version="1.0" encoding="UTF-8" ?>
<students
  xmlns="http://www.lagou.com/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>

</students>

```

4) 引入的命名空间跟哪个xsd文件对应?

使用schemaLocation来指定: 两个取值: 第一个为命名空间 第二个为xsd文件的路径

```

<?xml version="1.0" encoding="UTF-8" ?>
<students
  xmlns="http://www.lagou.com/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lagou.com/xml student.xsd"
>

</students>

```

5) 命名空间

指的是一个环境,所用的标签来自于哪个环境定义的。

6) student.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<students
  xmlns="http://www.lagou.com/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lagou.com/xml student.xsd"
>
  <student number="hehe_1234">
    <name>张百万</name>
    <age>25</age>
    <sex>female</sex>
  </student>

  <student number="hehe_0000">
    <name>小斌</name>
    <age>20</age>
    <sex>male</sex>
  </student>
</students>
```

4. XML 解析

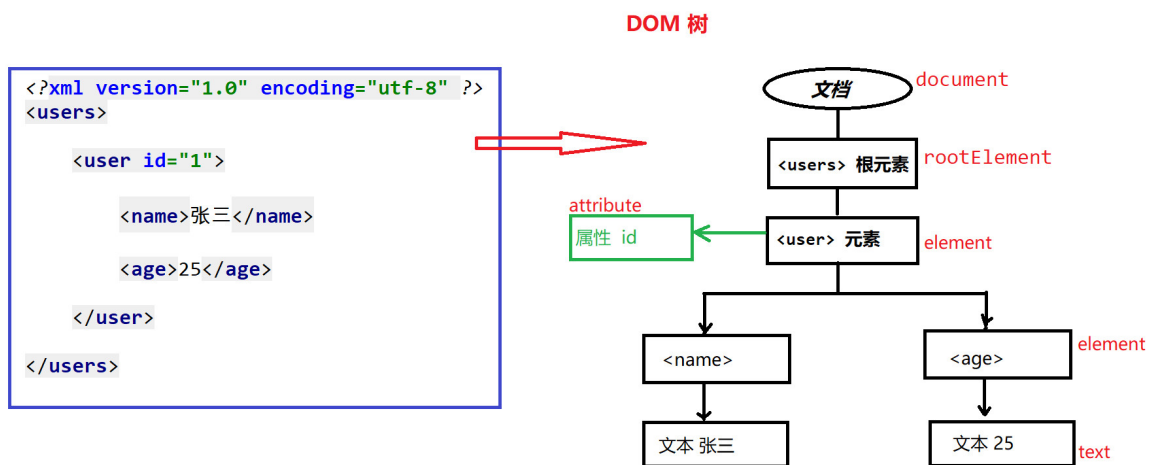
4.1 解析概述

当将数据存储在XML后,我们就希望通过程序获得XML的内容。如果我们使用Java基础所学习的IO知识是可以完成的,不过你需要非常繁琐的操作才可以完成,且开发中会遇到不同问题(只读、读写)。人们为不同问题提供不同的解析方式,并提交对应的解析器,方便开发人员操作XML。

4.2 XML解析方式

开发中比较常见的解析方式有两种,如下:

- DOM: 要求解析器把整个XML文档装载到内存,并解析成一个Document对象。
 - 优点: 元素与元素之间保留结构关系,故可以进行增删改查操作。
 - 缺点: XML文档过大,可能出现内存溢出显现。
- SAX: 是一种速度更快,更有效的方法。它逐行扫描文档,一边扫描一边解析。并以事件驱动的方式进行具体解析,每执行一行,都将触发对应的事件。(了解)
 - 优点: 占用内存少 处理速度快,可以处理大文件
 - 缺点: 只能读,逐行后将释放资源。



4.3 XML常见的解析器

解析器：就是根据不同的解析方式提供的具体实现。有的解析器操作过于繁琐，为了方便开发人员，有提供易于操作的解析开发包

- JAXP：sun公司提供的解析器，支持DOM和SAX两种思想
- **DOM4j**：一款非常优秀的解析器，Dom4j是一个易用的、开源的库，用于XML，XPath和XSLT。它应用于Java平台，采用了Java集合框架并完全支持DOM，SAX和JAXP。
- Jsoup：jsoup 是一款Java 的HTML解析器，也可以解析XML
- PULL：Android内置的XML解析方式，类似SAX。

4.4 dom4j 的使用

4.4.1 导入JAR包

 dom4j-1.6.1.jar	2013/2/27 18:17	Executable Jar File
---	-----------------	---------------------

4.4.2 API介绍

使用核心类SaxReader加载xml文档获得Document，通过Document 对象获得文档的根元素，然后就可以操作了

常用API如下：

- SaxReader对象
 - read(...) 加载执行xml文档
- Document对象
 - getRootElement() 获得根元素
- Element对象
 - elements(...) 获得指定名称的所有子元素。可以不指定名称
 - element(...) 获得指定名称的第一个子元素。可以不指定名称
 - getName() 获得当前元素的元素名
 - attributeValue(...) 获得指定属性名的属性值
 - elementText(...) 获得指定名称子元素的文本值
 - getText() 获得当前元素的文本内容

4.4.3 准备xml文件

编写user.xsd schema约束


```

<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns="http://www.lagou.com/xml"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://www.lagou.com/xml"
             elementFormDefault="qualified">

    <xsd:element name="users" type="usersType"/>
    <xsd:complexType name="usersType">
        <xsd:sequence>
            <xsd:element name="user" type="userType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="userType">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="age" type="ageType" />
            <xsd:element name="hobby" type="hobbyType" />
        </xsd:sequence>
        <xsd:attribute name="id" type="numberType" use="required"/>
    </xsd:complexType>

    <xsd:simpleType name="ageType">
        <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="100"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="hobbyType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="抽烟"/>
            <xsd:enumeration value="喝酒"/>
            <xsd:enumeration value="烫头"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="numberType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\d"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>

```

编写user.xml 引入约束

```

<?xml version="1.0" encoding="UTF-8" ?>
<users
    xmlns="http://www.lagou.com/xml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.lagou.com/xml user.xsd"
>
    <user id="1">
        <name>张百万</name>
        <age>20</age>
    </user>

```

```

        <hobby>抽烟</hobby>
    </user>

    <user id="2">
        <name>于谦</name>
        <age>50</age>
        <hobby>喝酒</hobby>
    </user>

    <user id="3">
        <name>刘能</name>
        <age>40</age>
        <hobby>烫头</hobby>
    </user>
</users>

```

4.4.4 读取XML

```

public class TestDOM4j {

    //获取XML文件中的 所有的元素名称(标签)
    @Test
    public void test1() throws DocumentException {

        //1. 获取XML解析对象
        SAXReader reader = new SAXReader();

        //2. 解析XML 获取 文档对象 document
        Document document =
reader.read("H:\\jdbc_work\\xml_task03\\src\\com\\lagou\\xml03\\user.xml");

        //3. 获取根元素
        Element rootElement = document.getRootElement();

        //获取根元素名称
        System.out.println(rootElement.getName());

        //获取 根元素下的标签
        List<Element> elements = rootElement.elements();
        for (Element element : elements) {
            System.out.println("根标签下的子节点: " + element.getName());

            List<Element> eList = element.elements();
            for (Element e : eList) {
                System.out.println("user标签下的子节点" + e.getName());
            }

            break;
        }
    }

    /**
     * 获取具体的节点内容 获取张百万的所有信息
     */
    @Test

```

```

public void test2() throws DocumentException {
    //1. 创建XML文档解析对象
    SAXReader sr = new SAXReader();

    //2. 读取XML获取到document对象
    Document document = sr.read("src\\com\\lagou\\xml02\\user.xml");

    //3. 获取根节点
    Element rootElement = document.getRootElement();

    //4. 得到当前节点的所有子节点
    List<Element> elements = rootElement.elements();

    //5. 获取第一个子节点
    Element user = elements.get(0);

    //6. 获取所有信息
    String id = user.attributeValue("id");
    String name = user.elementText("name");
    String age = user.elementText("age");
    //使用getText获取当前元素的文本内容
    String hobby = user.element("hobby").getText();

    //打印
    System.out.println(id+" " + name + " " + age + " " + hobby);
}
}

```

4.5 xpath方式读取xml

4.5.1 xpath介绍

XPath 是一门在 XML 文档中查找信息的语言。可以使用xpath查找xml中的内容。

XPath 的好处

由于DOM4J在解析XML时只能一层一层解析，所以当XML文件层数过多时使用会很不方便，结合XPATh就可以直接获取到某个元素

XPath 教程



简介

XPath由W3C的**XPath 1.0 标准**描述,本教程通过实例来展示XPath的一些特性.

你可以从以下内容开始:

- **实例 1**

1) 需要再导入 jaxen-1.1-beta-6.jar

名称	修改日期	类型	大小
endorsed	2005/5/16 14:28	文件夹	
test	2005/5/16 14:28	文件夹	
tools	2005/5/16 14:28	文件夹	
 jaxen-1.1-beta-6.jar	2005/5/16 14:28	Executable Jar File	239 KB
 jaxme-api-0.3.jar	2005/5/16 14:28	Executable Jar File	30 KB

4.5.2 XPath基本语法介绍

使用dom4j支持xpath的操作的几种主要形式

语法	说明
/AAA/DDD/BBB	表示一层一层的，AAA下面 DDD下面的BBB
//BBB	表示和这个名称相同，表示只要名称是BBB，都得到
//*	所有元素
BBB[1] , BBB[last()]	第一种表示第一个BBB元素, 第二种表示最后一个BBB元素
//BBB[@id]	表示只要BBB元素上面有id属性，都得到
//BBB[@id='b1']	表示元素名称是BBB,在BBB上面有id属性，并且id的属性值是b1

4.5.3 API介绍

2) 常用方法：

- selectSingleNode(query): 查找和 XPath 查询匹配的一个节点。
 - 参数是Xpath 查询串。
- selectNodes(query): 得到的是xml根节点下的所有满足 xpath 的节点；
 - 参数是Xpath 查询串。
- Node: 节点对象

4.5.4 Xpath读取XML

3) 数据准备 book.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<bookstore>
  <book id="book1">
    <name>金瓶梅</name>
    <author>金圣叹</author>
    <price>99</price>
  </book>
  <book id="book2">
    <name>红楼梦</name>
    <author>曹雪芹</author>
    <price>69</price>
  </book>
  <book id="book3">
    <name>Java编程思想</name>
    <author>埃克尔</author>
    <price>59</price>
  </book>
</bookstore>
```

```
</book>
</bookstore>
```

4) 代码示例

1. 使用selectSingleNode方法 查询指定节点中的内容

```
/*
 * 1. 使用selectSingleNode方法 查询指定节点中的内容
 * */
@Test
public void test1() throws DocumentException {

    //1.创建解析器对象
    SAXReader sr = new SAXReader();

    //2.获取文档对象
    Document document =
sr.read("H:\\jdbc_work\\xml_task03\\src\\com\\lagou\\xml03\\book.xml");

    //3.调用 selectSingleNode() 方法,获取name节点对象
    Node node1 = document.selectSingleNode("/bookstore/book/name");
    System.out.println("节点: " + node1.getName());
    System.out.println("书名: " + node1.getText());

    //4.获取第二本书的名称
    Node node2 = document.selectSingleNode("/bookstore/book[2]/name");
    System.out.println("第二本书的书名为: " + node2.getText());

}
```

2.使用selectSingleNode方法 获取属性值,或者属性值对应的节点

```
/*
 * 2.使用selectSingleNode方法 获取属性值,或者属性值对应的节点
 * */
@Test
public void test2() throws DocumentException {

    //1.创建解析器对象
    SAXReader sr = new SAXReader();

    //2.获取文档对象
    Document document =
sr.read("H:\\jdbc_work\\xml_task03\\src\\com\\lagou\\xml03\\book.xml");

    //3.获取第一个book节点的 id属性的值
    Node node1 = document.selectSingleNode("/bookstore/book/attribute::id");
    System.out.println("第一个book的id值为: " + node1.getText());

    //4.获取最后一个book节点的 id属性的值
```

```

        Node node2 =
document.selectSingleNode("/bookstore/book[last()]/attribute::id");
        System.out.println("最后一个book节点的id值为: " + node2.getText());

//5.获取id属性值为 book2的 书名
Node node3 = document.selectSingleNode("/bookstore/book[@id='book2']");
String name = node3.selectSingleNode("name").getText();
System.out.println("id为book2的书名是: " + name);

    }

```

3. 使用 selectNodes()方法 获取对应名称的所有节点

```

/*
 * 3.使用 selectNodes()方法 获取对应名称的所有节点
 *
 * */
@Test
public void test3() throws DocumentException {

    //1.创建解析器对象
    SAXReader sr = new SAXReader();

    //2.获取文档对象
    Document document =
sr.read("H:\\jdbc_work\\xml_task03\\src\\com\\lagou\\xml03\\book.xml");

    //3.获取所有节点,打印节点名
    List<Node> list = document.selectNodes("/*");
    for (Node node : list) {
        System.out.println("节点名: " + node.getName());
    }

    //4.获取所有的书名
    List<Node> names = document.selectNodes("//name");
    for (Node name : names) {
        System.out.println(name.getText());
    }

    //5.获取指定 id值为book1的节点的所有 内容
    List<Node> book1 =
document.selectNodes("/bookstore/book[@id='book1']/*");
    for (Node node : book1) {
        System.out.println(node.getName()+" = " + node.getText());
    }
}

```

5. JDBC自定义XML

5.1 定义配置文件

1) 创建自定义xml 文件, 保存 数据库连接信息

jdbc-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<jdbc>
  <property name="driverClass">com.mysql.jdbc.Driver</property>
  <property name="jdbcUrl">jdbc:mysql://localhost:3306/db5?
characterEncoding=UTF-8</property>
  <property name="user">root</property>
  <property name="password">123456</property>
</jdbc>
```

5.2 编写工具类(配置式)

2) 编写工具类,使用xpath 读取数据库信息

```
public class JDBCUtils {

    //1. 定义字符串变量, 记录获取连接所需要的信息
    public static String DRIVERNAME;
    public static String URL;
    public static String USER;
    public static String PASSWORD;

    //2. 静态代码块
    static {
        try {
            //使用 xpath读取 xml中的配置信息
            SAXReader sr = new SAXReader();
            Document document =
sr.read("H:\\workspace01\\JDBC_day02\\src\\com\\lagou\\xml03\\jdbc-config.xml");

            Node node =
document.selectSingleNode("/jdbc/property[@name='driverClass']");
            //System.out.println(node.getText());
            DRIVERNAME = node.getText();
            URL =
document.selectSingleNode("/jdbc/property[@name='jdbcUrl']").getText();
            USER =
document.selectSingleNode("/jdbc/property[@name='user']").getText();
            PASSWORD =
document.selectSingleNode("/jdbc/property[@name='password']").getText();

            //注册驱动
            Class.forName(DRIVERNAME);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //3. 获取连接的静态方法
    public static Connection getConnection(){

        try {
            //获取连接对象
            Connection connection = DriverManager.getConnection(URL, USER,
PASSWORD);
```

```

        //返回连接对象
        return connection;

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
}

```

5.3 测试工具类

3) 测试：获取所有员工的姓名

```

//获取所有员工的姓名
public static void main(String[] args) {

    try {
        //1.获取连接
        Connection connection = JDBCUtils.getConnection();

        //2.获取 statement ,执行SQL
        Statement statement = connection.createStatement();
        String sql = "select * from employee";

        //3.处理结果集
        ResultSet resultSet = statement.executeQuery(sql);
        while(resultSet.next()){
            String ename = resultSet.getString("ename");
            System.out.println(ename);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```