Xml解析

# 文件

## 文件格式

1. 二进制格式

* 概述

按照特定的编码规则进行编写的文件内容。如：word

* 优点
* 处理速度快，更容易被计算机所理解
* 表现内容丰富(表格、图片、音频等)
* 指定内容的格式
* 缺点

必须用指定软件打开

1. 文本格式

* 概述

本质上还是二进制的内容。文本格式会按照指定的编码格式进行存储。编码格式：ASCII、GBK、Unicode等

* 优点
* 可以通过任意软件进行打开和使用
* 具有较强的兼容性
* 缺点

内容格式单一

# 标记性语言

## SGML

复杂、难以使用

## Html

1. 作用

以特定的格式对数据进行表现(表现数据)

1. 优点

* 表现内容丰富
* Html使用简单

1. 缺点

无法持久化的保存数据

## Xml

1. 概述

Extensable Markup Language的缩写。Xml不是用来替代Html的，而是对Html的一个补充。Xml主要用来持久化存储数据。

1. 作用

用来进行持久化的数据存储。

1. Html和Xml

* Html中的标签是预定义的；xml的标签是自定义的
* Html中的标签可以没有结束标签(自封闭标签);Xml中的标签必须是成对出现的
* Html中的标签不区分大小写；Xml中的标签是区分大小写的。
* Html侧重数据的表现；Xml主要用来存储数据

1. Xml格式

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <Students>  <student>  <name>Mike</name>  <age>20</age>  </student>  <student>  <name>Joe</name>  <age>23</age>  </student>  </students> |

1. 优点

* 具有平台无关性
* 传输效率高

1. 注意事项

* 文档声明必须是文件的第一行
* 一个文档只能有一个根节点
* 节点都是成对出现的
* Xml节点的名称是严格区分大小的
* 不能交叉定义，必须一一匹配
* 节点名称不能以数字开头
* 属性值必须用””引起来

1. 示例

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <students>  <student>  <name>Mike</name>  <age>20</age>  </student>  <student>  <name>Joe</name>  <age>23</age>  </student>  </students> |
| <?xml version="1.0" encoding="utf-8" ?>  <students>  <student name="Mike" age="20"></student>  <student name="Joe" age="23"></student>  </students> |

1. <![CDATA[]]>

* 格式

<![CDATA[内容]]>

* 作用

当内容与文件的语法格式发生冲突时，使用<![ CDATA []]可以避免编译器对内容进行解析。

* 示例

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <students>  <info>  <a href="www.baidu.com">百度</a>  </info>  <info>20>200</info>  <info>他的结束标签是&lt;/info&gt;</info>  <info>  <![CDATA[他的结束标签是</info>]]>  </info>  </students> |

1. Xml解析

* Dom
* 概述

Dom方式是一组API，他提供了用于进行Xml文件读写的相关方法。

* 示例

|  |
| --- |
| /\*\*  \* 使用Dom方式创建Xml文件  \* **@param** path  \* **@throws** ParserConfigurationException  \* **@throws** TransformerException  \* **@throws** FileNotFoundException  \*/  **public** **void** create(String path) **throws** ParserConfigurationException, TransformerException, FileNotFoundException{  //1、定义一个DocumentDuilderFactory对象  DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();    //2、获取DocumentBuilder对象  DocumentBuilder builder = factory.newDocumentBuilder();  //3、获取dom对象实例  Document doc = builder.newDocument();    //4、创建元素  //1)、创建根元素  Element root = doc.createElement("Users");    //2)、创建name元素  //a、创建节点元素  Element name = doc.createElement("Name");  //b、为节点元素添加内容  name.appendChild(doc.createTextNode("Mike"));  //c、将节点元素挂接到指定节点下  root.appendChild(name);  //3)、创建age元素  Element age = doc.createElement("Age");  age.appendChild(doc.createTextNode("20"));  root.appendChild(age);  //4)、根节点挂接到文档对象  doc.appendChild(root);    //5、创建TransformerFactory对象  TransformerFactory formerFactory= TransformerFactory.*newInstance*();    //6、创建Transformer对象  Transformer former = formerFactory.newTransformer();  //设置文件属性  former.setOutputProperty("encoding", "ascii");    //7、构建DOMSource对象  DOMSource dom = **new** DOMSource(doc);    //8、创建StreamResult对象  PrintWriter writer= **new** PrintWriter(**new** File(path));  StreamResult result = **new** StreamResult(writer);    //9、将Result源转为结果  former.transform(dom, result);  System.*out*.println("Dom方式创建完成......");  } |
| /\*\*  \* 以Dom方式读取Xml文档的内容  \* **@param** path  \* **@throws** ParserConfigurationException  \* **@throws** SAXException  \* **@throws** IOException  \*/  **public** **void** read(String path) **throws** ParserConfigurationException, SAXException, IOException{  //1、创建DocumentBuilderFactory  DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();    //2、创建DocumentBuilder对象  DocumentBuilder builder = factory.newDocumentBuilder();    //3、创建文档对象  //通过builder对象的parse()方法将文件转换并加载到内存中  Document doc = builder.parse(path);    //4、解析Xml文档  NodeList nodes = doc.getChildNodes();    //判断是否加载到节点  **if**(nodes.getLength()<1) **return**;    //获取根节点的所有子节点  //nodes.item(0) --- 获取根节点  NodeList list = nodes.item(0).getChildNodes();  //遍历节点的集合  **for**(**int** i = 0;i<list.getLength();i++){  System.*out*.println(list.item(i).getNodeName() + " | " + list.item(i).getTextContent());  }    } |

* 缺点

一次性将文档加载到内存中，比较浪费资源。

* 适用

一次加载，多次访问的情况下使用

* Jdom
* 概述

Dom及Sax方式代码过长，所以在Dom和Sax基础上发展出JDom解析方式。所以说JDom是以Dom和Sax为基础进行实现的

* 示例

|  |
| --- |
| /\*\*  \* JDOm方式创建Xml文档  \*  \* **@param** path  \* **@throws** FileNotFoundException  \* **@throws** IOException  \*/  **public** **void** create(String path) **throws** FileNotFoundException, IOException{  //1、创建Document对象  Document doc = **new** Document();    //2、创建element对象  //1)、创建根节点对象  Element root = **new** Element("User");  //将根节点添加到文档中  doc.addContent(root);    //2)、创建子节点对象  //a、创建节点对象  Element name = **new** Element("Name");  //b、设置节点的文本内容  name.setText("Joe");  //将子节点添加到root根节点下  root.addContent(name);    Element age = **new** Element("age");  age.setText("22");  root.addContent(age);    //3、获取Format对象  Format formatter = Format.*getPrettyFormat*();  //设置编码格式  formatter.setEncoding("utf-8");  //设置缩进距离  //一般在网络环境下不建议使用----缩进也是字符  formatter.setIndent(" ");    //4、创建XMLOutputter对象  //通过该对象对内容进行输出  XMLOutputter outputer = **new** XMLOutputter(formatter);  outputer.output(doc, **new** FileOutputStream(path));    System.*out*.println("JDom方式创建结束......");  } |
| /\*\*  \* 以JDom方式解析Xml文档  \* **@param** path  \* **@throws** JDOMException  \* **@throws** IOException  \*/  **public** **void** read(String path) **throws** JDOMException, IOException{  //1、构建SAXBuilder对象  SAXBuilder builder = **new** SAXBuilder();    //2、通过SaxBuilder对象获取Document对象  Document doc = builder.build(**new** File(path));    //3、解析Xml文档  //获取根节点元素  Element root = doc.getRootElement();    //4、获取所有子节点  List<Element> list = root.getChildren();    //5、便利所有子节点  **for**(Element ele : list){  System.*out*.println(ele.getName() + " | " + ele.getText());  }  } |

* 优点

代码量较少，更符合80-20原则

* 适用

代码量较小的解析过程

* Dom4j
* 概述

Dom4j提供了一组开源、高效的Xml操作的API，sun公司部分代码中也在使用Dom4j方式进行呢Xml解析

* 示例

|  |
| --- |
| /\*\*  \* 使用Dom4j方式解析Xml生成Xml文档  \*  \* **@param** path  \* **@throws** IOException  \*/  **public** **void** create(String path) **throws** IOException{  //1、通过DocumentHelper对象构建Document对象  Document doc = DocumentHelper.*createDocument*();    //2、创建节点  //创建根节点  Element root = doc.addElement("Users");    //创建name子节点  Element name = root.addElement("Name");  name.setText("Tom");    //创建Age子节点  Element age = root.addElement("Age");  age.setText("30");    //3、创建Xml文件  Writer writer = **new** PrintWriter(path);  //创建XMLWriter对象  XMLWriter xw = **new** XMLWriter(writer);  //通过XMLWriter生成Xml文档  xw.write(doc);  writer.close();    System.*out*.println("Dom4j方式生成成功......");  } |
| /\*\*  \* Dom4j方式读取Xml文件  \*  \* **@param** path  \* **@throws** DocumentException  \*/  **public** **void** read(String path) **throws** DocumentException{  //1、获取SAXReader对象  SAXReader reader = **new** SAXReader();    //2、获取Document对象  Document doc = reader.read(**new** File(path));    //3、读取节点  //1)、获取根节点对象  Element root = doc.getRootElement();    //2)、获取根节点的子节点  Iterator<Element> iterator = root.elementIterator();  //通过迭代器遍历节点集合  **while**(iterator.hasNext()){  //获取节点集合  Element node = iterator.next();  System.*out*.println(node.getName() + " | " + node.getText());  }  } |

* Xpath
  + 概述

Xpath是使用路径表达式进行节点查找的一种方式。

* + 特殊符号

/ ：表示绝对路径

// ：表示选择所有满足“//”之后的条件的所有节点（不考虑层级问题 ）

\* ：表示所有元素

[] ：表示选取集合指定的元素

@ ：表示使用特定的属性

* + 使用

Dom4J支持Xpath查询。

* + 示例

|  |
| --- |
| /\*\*  \* 使用Xpath方式读取节点内容  \* Dom4j方式同样支持Xpath  \* 需要导入jaxen-1.1-beta-6.jar  \*  \* **@param** path  \* **@throws** DocumentException  \*/  **public** **void** Xpath(String path) **throws** DocumentException{  //1、获取SAXReader对象  SAXReader reader = **new** SAXReader();  //2、获取Document对象  Document doc = reader.read(**new** File(path));  //3、通过Xpath获取节点对象  Node node = doc.selectSingleNode("/Users/Student/Name");  //4、操作节点  System.*out*.println(node.getName() + " | " + node.getText());  } |

* + 注意事项

需要导入jaxen-1.1-beta-6.jar

* 应用
  + 配置文件

Jsp的配置文件

Hibernate的配置文件

Tomcat的配置文件

* + 文件传输

Ajax

WebService

* + 持久化数据存储