SqlSession

1 用于操作增删改查

2 用户获取连接对象Connection

3 获取配置信息Configuration

4 Configuration中存入了mapper的所有配置文件，通过mapperid查找对应的sql

具体一条sql在mapper的表示

|  |
| --- |
| MapperStatement  作用: 代表一个Mapper.xml文件中的一个sql标签    RawSqlSource  作用：原始SQL，即具体的sql语句    BoundSql  作用:解析的具体SQL |

XML解析

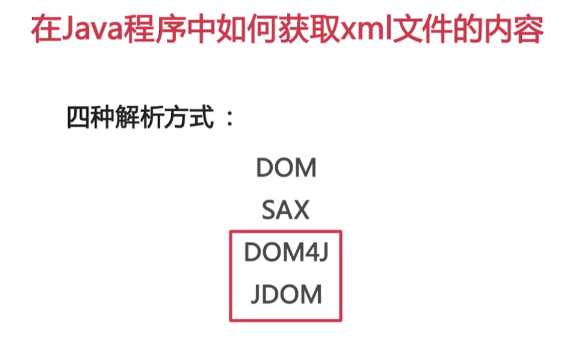
Mybatis中Dom解析api分析

该剖析有利于更好的理解mybatis的设计，同时也有助于spring对于xml容器的处理

1 首先熟知，如何定义和解析xml文件

2 xml文件解析的几种方式

Xml解析的工具



3 对比哪种好

类加载器

1 作用：加载不同类型下的class文件到jvm中，这里需要理解不同类加载加载的class不同，jre目录下需要系统类加载器、etx目录下需要扩展类加载器、classpath目录下需要应用类加载器，所以需要明白的是指定一个class文件路径后，需要各加载器遍历查找

2 分类：系统类加载器、扩展类加载器、应用类加载器、自定义类加载

3 类加载的获取

通过Class文件对象获取：因每个类都对应一个唯一的Class文件类，每个Class文件类都会绑定一个对应的类加载器。

ClassLoader类加载器对象

|  |
| --- |
| \*.class.getClassLoader();  # 类形式获取  Class.**class**.getClassLoader(); |

通过类加载器可以获取资源路径下的文件，此时需要使用各加载器循环遍历处理。因为指定的class文件不知道具体位置，可能在jre、etx、classpath下

Resource 定义资源加载器

作用: 通过类加载器获取资源路径

原理：利用不同类型的类加载器进行遍历查找

加载器分类：

1 系统类加载

ClassLoader.getSystemClassLoader();

查找jar文件

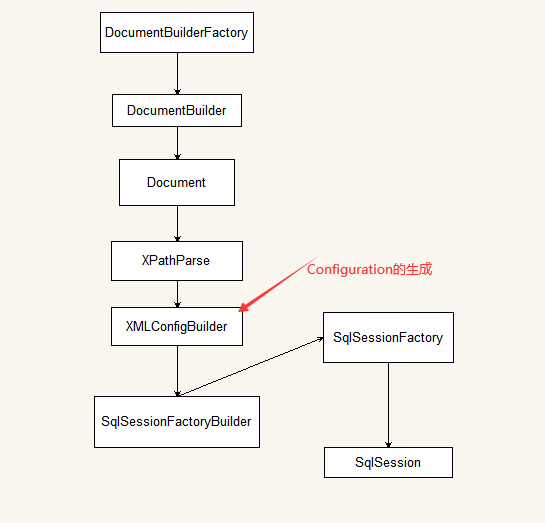
1 根据包名查找该包名下的jar文件。Class文件

SqlSessionFactoryBuilder ： 用于生成SqlSessionFactory

SqlSessionFactory : 用于生成SqlSession

SqlSession ： 操作数据的回话对象

调动接口图



SqlSessionFactoryBuilder

SqlSessionFactory

SqlSession

XMLConfigBuilder

（生成Configuration对象）

XPathParse

DocumentBuilderFactory

Document

DocumentBuilder

通用DocumentAPI图

XPath

DocumentBuilder

Document

DocumentBuilderFactory

直接上代码

|  |
| --- |
| **private** Document createDocument(InputSource inputSource) {  *// important: this must only be called AFTER common constructor* **try** {  DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  factory.setFeature(XMLConstants.***FEATURE\_SECURE\_PROCESSING***, **true**);  *// 指定此代码生成的解析器将在文档解析时验证文档 默认false不校验* factory.setValidating(**validation**);  *// 指定此代码生成的解析器将为XML命名空间提供支持。 默认情况下，该值设置为false* factory.setNamespaceAware(**false**);  *// 指定此代码生成的解析器将忽略注释。 默认情况下，该值设置为false 。* factory.setIgnoringComments(**true**);  *// 指定是否解析器必须在解析XML文档时消除元素内容中的空格* factory.setIgnoringElementContentWhitespace(**true**);  *// 解析器将CDATA节点转换为文本节点并将其附加到相邻（如果有的话）文本节点* factory.setCoalescing(**false**);  *// 指定此代码生成的解析器将扩展实体引用节点。* factory.setExpandEntityReferences(**true**);   DocumentBuilder builder = factory.newDocumentBuilder();  *// 解析要解析的XML文档中存在的实体的EntityResolver* builder.setEntityResolver(**entityResolver**);  builder.setErrorHandler(**new** ErrorHandler() {  **@Override  public void** error(SAXParseException exception) **throws** SAXException {  **throw** exception;  }   **@Override  public void** fatalError(SAXParseException exception) **throws** SAXException {  **throw** exception;  }   **@Override  public void** warning(SAXParseException exception) **throws** SAXException {  *// NOP* }  });  **return** builder.parse(inputSource);  } **catch** (Exception e) {  **throw new** BuilderException(**"Error creating document instance. Cause: "** + e, e);  } } |

上面是获取document对象

|  |
| --- |
| XPathFactory xPathFactory = XPathFactory.*newInstance*(); XPath xPath = xPathFactory.newXPath(); **try**{  // 获取根元素  Node root= (Node) xPath.evaluate(**"/root"**, document, XPathConstants.***NODE***);  System.***out***.println(**"settings: "** + root.getNodeName()); }**catch** (XPathExpressionException xp){  } |

XPath

**XPath 是一门在 XML 文档中查找信息的语言(API)。XPath 用于在 XML 文档中通过元素和属性进行导航**

## **什么是 XPath?**

* XPath 使用路径表达式在 XML 文档中进行导航
* XPath 包含一个标准函数库
* XPath 是 XSLT 中的主要元素
* XPath 是一个 W3C 标准

## **XPath 路径表达式**

XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

## **XPath 标准函数**

XPath 含有超过 100 个内建的函数。这些函数用于字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值等等。

## **XPath 节点**

**在 XPath 中，有七种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档节点（或称为根节点）。**

## **XPath 术语**

### **节点（Node）**

在 XPath 中，有七种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点。XML 文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。

## **XPath 语法**

**XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是通过沿着路径 (path) 或者步 (steps) 来选取的**

## **XML 实例文档**

我们将在下面的例子中使用这个 XML 文档。

<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>

<title lang="eng">Harry Potter</title>

<price>29.99</price>

</book>

<book>

<title lang="eng">Learning XML</title>

<price>39.95</price>

</book>

</bookstore>

## **选取节点**

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。

### **下面列出了最有用的路径表达式：**

|  |  |
| --- | --- |
| **表达式** | **描述** |
| nodename | 选取此节点的所有子节点。 |
| / | 从根节点选取。 |
| // | 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。 |
| . | 选取当前节点。 |
| .. | 选取当前节点的父节点。 |
| @ | 选取属性。 |

### **实例**

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| bookstore | 选取 bookstore 元素的所有子节点。 |
| /bookstore | 选取根元素 bookstore。  注释：假如路径起始于正斜杠( / )，则此路径始终代表到某元素的绝对路径！ |
| bookstore/book | 选取属于 bookstore 的子元素的所有 book 元素。 |
| //book | 选取所有 book 子元素，而不管它们在文档中的位置。 |
| bookstore//book | 选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。 |
| //@lang | 选取名为 lang 的所有属性。 |

## **谓语（Predicates）**

谓语用来查找某个特定的节点或者包含某个指定的值的节点。

谓语被嵌在方括号中。

### **实例**

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| /bookstore/book[1] | 选取属于 bookstore 子元素的第一个 book 元素。 |
| /bookstore/book[last()] | 选取属于 bookstore 子元素的最后一个 book 元素。 |
| /bookstore/book[last()-1] | 选取属于 bookstore 子元素的倒数第二个 book 元素。 |
| /bookstore/book[position()<3] | 选取最前面的两个属于 bookstore 元素的子元素的 book 元素。 |
| //title[@lang] | 选取所有拥有名为 lang 的属性的 title 元素。 |
| //title[@lang='eng'] | 选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。 |
| /bookstore/book[price>35.00] | 选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。 |
| /bookstore/book[price>35.00]/title | 选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。 |

## **选取未知节点**

XPath 通配符可用来选取未知的 XML 元素。

|  |  |
| --- | --- |
| **通配符** | **描述** |
| \* | 匹配任何元素节点。 |
| @\* | 匹配任何属性节点。 |
| node() | 匹配任何类型的节点。 |

### **实例**

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| /bookstore/\* | 选取 bookstore 元素的所有子元素。 |
| //\* | 选取文档中的所有元素。 |
| //title[@\*] | 选取所有带有属性的 title 元素。 |

## **选取若干路径**

通过在路径表达式中使用“|”运算符，您可以选取若干个路径。

### **实例**

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

|  |  |
| --- | --- |
| **路径表达式** | **结果** |
| //book/title | //book/price | 选取 book 元素的所有 title 和 price 元素。 |
| //title | //price | 选取文档中的所有 title 和 price 元素。 |
| /bookstore/book/title | //price | 选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。 |

## **XML实例文档**

我们将在下面的例子中使用这个 XML 文档：

### **"books.xml" :**

<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">

<title lang="en">Everyday Italian</title>

<author>Giada De Laurentiis</author>

<year>2005</year>

<price>30.00</price>

</book>

<book category="CHILDREN">

<title lang="en">Harry Potter</title>

<author>J K. Rowling</author>

<year>2005</year>

<price>29.99</price>

</book>

<book category="WEB">

<title lang="en">XQuery Kick Start</title>

<author>James McGovern</author>

<author>Per Bothner</author>

<author>Kurt Cagle</author>

<author>James Linn</author>

<author>Vaidyanathan Nagarajan</author>

<year>2003</year>

<price>49.99</price>

</book>

<book category="WEB">

<title lang="en">Learning XML</title>

<author>Erik T. Ray</author>

<year>2003</year>

<price>39.95</price>

</book>

</bookstore>

## **加载 XML 文档**

所有现代浏览器都支持使用 XMLHttpRequest 来加载 XML 文档的方法。

针对大多数现代浏览器的代码：

var xmlhttp=new XMLHttpRequest()

针对古老的微软浏览器（IE 5 和 6）的代码：

var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")

## **选取节点**

不幸的是，Internet Explorer 和其他处理 XPath 的方式不同。

在我们的例子中，包含适用于大多数主流浏览器的代码。

Internet Explorer 使用 selectNodes() 方法从 XML 文档中的选取节点：

xmlDoc.selectNodes(*xpath*);

Firefox、Chrome、Opera 以及 Safari 使用 evaluate() 方法从 XML 文档中选取节点：

xmlDoc.evaluate(*xpath*, xmlDoc, null, XPathResult.ANY\_TYPE,null);

## **选取所有 title**

下面的例子选取所有 title 节点：

/bookstore/book/title

|  |
| --- |
| **<html>**  **<body>**  **<script type="text/javascript">**  **function loadXMLDoc(dname)**  **{**  **if (window.XMLHttpRequest)**  **{**  **xhttp=new XMLHttpRequest();**  **}**  **else**  **{**  **xhttp=new ActiveXObject("Microsoft.XMLHTTP");**  **}**  **xhttp.open("GET",dname,false);**  **xhttp.send("");**  **return xhttp.responseXML;**  **}**  **xml=loadXMLDoc("/example/xmle/books.xml");**  **path="/bookstore/book/title"**  **// code for IE**  **if (window.ActiveXObject)**  **{**  **var nodes=xml.selectNodes(path);**  **for (i=0;i<nodes.length;i++)**  **{**  **document.write(nodes[i].childNodes[0].nodeValue);**  **document.write("<br />");**  **}**  **}**  **// code for Mozilla, Firefox, Opera, etc.**  **else if (document.implementation && document.implementation.createDocument)**  **{**  **var nodes=xml.evaluate(path, xml, null, XPathResult.ANY\_TYPE, null);**  **var result=nodes.iterateNext();**  **while (result)**  **{**  **document.write(result.childNodes[0].nodeValue);**  **document.write("<br />");**  **result=nodes.iterateNext();**  **}**  **}**  **</script>**  **</body>**  **</html>** |

**输出结果**

|  |
| --- |
| Harry Potter Everyday Italian Learning XML XQuery Kick Start |