# JAXP 全部内容，第 2 部分

**使用 Sun 的 XML 处理工具包来转换 XML**

在这个由两部分组成的系列文章中，第 1 部分介绍了 Java™ API for XML Processing (JAXP) 及其解析和验证特性。JAXP 还为 Java 程序员提供了使用可扩展样式表语言 (XSL) 转换 XML 文档的能力。通过直接编程访问和 XSL 模板，JAXP 使得从一种 XML 格式转换为另一种格式非常容易。本文展示如何使用 JAXP 来转换 XML 文档和如何高速缓存 XSL 样式表以实现最佳性能。

0http://dw1.s81c.com/developerworks/i/v17/dw-cmts-arrow.png [评论](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#icomments)

[Brett McLaughlin](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#authorN10028) ([brett@newInstance.com](mailto:brett@newInstance.com?subject=JAXP%20%E5%85%A8%E9%83%A8%E5%86%85%E5%AE%B9%EF%BC%8C%E7%AC%AC%202%20%E9%83%A8%E5%88%86&cc=dwxed@us.ibm.com)), 作家和编辑, O'Reilly Media, Inc.

2005 年 6 月 13 日

* [expand](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#toggle)**内容**



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

[**开始您的试用**](https://developer.ibm.com/sso/bmregistration?lang=zh_CN&ca=dwchina-_-bluemix-_-x-jaxp2-_-sidebar)

在 JAXP 的早期版本中，该首字母缩写代表 Java API for XML Parsing。在 [第 1 部分](http://www.ibm.com/developerworks/cn/xml/x-jaxp/) 中了解到，JAXP 是位于 SAX 和 DOM 之上的层，它允许 Java 程序员执行开发商中立的 XML 解析。最初，这是 JAXP 的全部特性。不过俗话说的好，过去是过去，现在是现在。

过去，Java 和 XML 组合本身主要用于解析。Java 应用程序只需读入 XML 文档，然后按程序处理文档的数据。但随着 XML 消费应用程序流行起来，很显然，各种应用程序所执行的操作有许多重叠。对于所有优秀的软件，重叠将导致规范（而且每次都产生新的有用的 API）。

XML 的广泛使用所产生的第一个规范是 XSL（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）。应用程序不断提取 XML 数据，添加一些格式化，然后将其显示在用户界面上——通常是作为 HTML、XHTML 或 WML。XSL 完成此任务，并在其上构建规范，从而允许应用程序抛弃其所有的专用转换代码。XSL 规范产生之后，XML 转换 API（Transformation API for XML，TrAX）随之出现（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）。TrAX 提供了在 Java 应用程序中使用 XSL 的简单一致的方法。目前，JAXP——相当长的链（和介绍）中的最后一链——已经将 TrAX 合并到核心 Java 开发环境中。按照所有的发展，以及最近的添加（比如扩展验证和 XPath 支持），JAXP 现在代表 Java API for XML Processing。本文重点介绍使用 JAXP 进行处理而非解析。

## 由此及彼

理解 XSL 的基本程序流对于掌握 JAXP 如何处理转换是非常关键的。如果对 XSL 十分陌生，则需要快速回顾一下 XSL 基本概念。即使您是 XSL 专家也要忍受我如此絮叨。

### 来源 (XML)

使用 XSL 时，必须从 XML 开始。我知道这听起来理所当然，但还是值得说明一下。您可能习惯以 XML 文件开始（比如 phonebook.xml），并将其传递到 XSL 处理器中。JAXP 不仅允许您传递文件，它还允许您做好多事，在下一节 [输入和输出](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#inout) 中将学习相关内容。

### 样式表 (XSL)

可能吸引大多数设计人员的是 XSL 样式表。样式表是一个指令集合，它指定特定类型的数据作为输入，并指定其他一组数据和格式化作为输出。但是切记，样式表应该对入站 XML 的结构进行操作，而不是对文档中的特定数据进行操作。这就确保样式表处理任何给定格式的 XML，而非特定的实例文档。

### 目标 (\*ML)

最后需要记住，只能从 XSL 中输出格式良好的标记语言。不能输出 Microsoft Word 文档或 PDF。一定要使用标记语言，比如 XML、XHTML、WML 或其他良好的 \*ML（标记语言）变种。

当然，对此我听到异议——已经看到了从 XML 输出 PDF 的应用程序，或将 XML 转换为 Excel 的应用程序。而且，可以接受特定格式的 XML 并将其转换为二进制格式的引擎确实存在。 但这并不属于 XSL 的领域；它是转换后处理。JAXP 会帮助转换 XML，但它不允许转换为二进制格式。

[**回页首**](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 输入和输出

通过简单的回顾可能已经了解到，许多 XML 转换只是关于输入和输出的。导入 XML，对它进行操作，然后输出 \*ML。在处理所有中间位（我意识到这是最有趣的地方）之前，将展示如何将数据输入到 JAXP 和如何将其输出返回。

### JAXP 的灵活性

JAXP 不是一般的转换引擎。它不能将 Java 属性文件或一次性文本格式转换为 XML 或其他格式的标记语言。事实上，JAXP 甚至不接受 HTML，除非它是格式良好的（因此产生了一些 XHTML）。所以在试图使 JAXP 成为更一般的 API 之前，应该意识到必须使用格式良好的 XML，否则其他一切都没用。

JAXP 的灵活性在于可以如何表示 XML。显而易见的原因是，JAXP 可以将 XML 接受为文件或包装文件的流。但它还可以将输入接受为 DOMDocument（这表示不可能存在于磁盘上的 XML 文档），或者接受为一系列 SAX 事件（这也表示 XML 文档）。使用这个附加的灵活性，可以将 JAXP 插入到任何 XML 事件链中。例如，如果具有一段代码，该代码使用 SAX 读入 XML 文档，然后将该数据的特定部分传递给另一个应用程序，则可以简单地在 SAX 代码和其他应用程序组件之间插入 JAXP。它可以消费 SAX 事件，按需转换数据，并将结果传递给接收应用程序组件。

这个故事的寓意是，可以以任何标准格式将 XML 传递给 JAXP，并以同样多的格式将其输出。即使目前不需要这种灵活性，或无法想像怎么可能使用所有这些不同格式，但是当您的需要变得更高级时，JAXP 已经准备好为您服务了。

### 输入来源

## 接口编程 101

如果使用接口对您来说相当陌生，则要注意，[清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 总是把接口放在等号 (=) 左边，把特定实现类放在右边。所以 Source 在左边，StreamSource、SAXSource 等实现类在右边。

javax.xml.transform.Source 接口是 JAXP 的所有输入和转换 API 的基础。该接口只定义了两个方法——getSystemId() 和 setSystemId(String systemId)。实际上，您将不会像处理 JAXP 提供的具体实现那样，过多地直接处理该接口：

* **javax.xml.transform.dom.DOMSource** 将 DOM Node（及其孩子）传递给 JAXP。
* **javax.xml.transform.sax.SAXSource** 将 SAX 回调结果（来自 XMLReader）传递给 JAXP。
* **javax.xml.transform.stream.StreamSource** 将包装在 File、InputStream 或 Reader 中的 XML 传递给 JAXP。

清单 1 展示了几种用于创建转换中使用的 Source 的方法：

##### 清单 1. 使用 Source 接口的实现

// Create a Source from a file on disk

Source fileSource =

new StreamSource(new File("phonebook.xml"));

// Create a Source from a DOM tree

Document myDomDocument = getDocument();

Source domSource = new DOMSource(myDomDocument);

// Create a Source from an InputStream

BufferedInputStream bis =

new BufferedInputStream(getInputStream());

Source streamSource = new StreamSource(bis);

// Create a Source from a reader and SAX InputSource

XMLReader myXMLReader = getXMLReader();

InputSource myInputSource = getInputSource();

Source saxSource = new SAXSource(myXMLReader, myInputSource);

[清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 几乎是自解释的。一旦获得 Source，就可以将 XML 输入 JAXP 的 XSL 处理部分。

### 输出结果

在讲述转换本身之前，将简单介绍一下 Source 的输出对应物——javax.xml.transform.Result。它甚至具有与 Source 相同的两个基本方法——getSystemId() 和 setSystemId(String systemId)。

与其输入对应物一样，一般使用 JAXP 的具体 Result 实现：

* **javax.xml.transform.dom.DOMResult** 将转换后的内容传递到 DOM Node 中。
* **javax.xml.transform.sax.SAXResult** 将转换的结果传递到 SAX ContentHandler 中。
* **javax.xml.transform.stream.StreamResult** 将转换后的 \*ML 传递到 File、OutputStream 或 Writer 中。

清单 2 展示了一些简单的例子，与 [清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code1) 中 Source 的例子十分相似：

##### 清单 2. 使用 Result 接口的实现

// Write to a file on disk

Result fileResult =

new StreamResult(new File("output.xml"));

// Write a Result to a DOM tree (inserted into the supplied Document)

Document myDomDocument = getDocument();

Result domResult = new DOMResult(myDomDocument);

// Create a Result from an OutputStream

BufferedOutputStream bos =

new BufferedOutputStream(getOutputStream());

Result streamResult = new StreamResult(bos);

// Create a Result to write to a SAX ContentHandler

ContentHandler myContentHandler = new MyContentHandler();

Result saxResult = new SAXResult(myContentHandler);

一旦理解了 Source 和 Result 接口以及与 JAXP 绑定在一起的实现之后，就差不多已经掌握了 XML 转换。

[**回页首**](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 使用 JAXP 执行转换

如果阅读 [第 1 部分](http://www.ibm.com/developerworks/xml/library/x-jaxp/?S_TACT=105AGX52&S_CMP=cn-a-x) 距今有一段时间了，或者谈到 JAXP 和解析时仍有些生疏，就应该花时间回顾一下 SAXParserFactory 和DOMBuilderFactory 类。您将发现，如果知道如何使用这些类，就已经能够完全理解 JAXP 转换如何工作了。

### 获得工厂

转换如此简单，以至于有点微不足道。首先，需要设置输入和输出接收器。将 Source 包装在输入 XML 文档和 XSL 样式表中。然后，创建一个接收器，以写入转换后的结果——然后将其包装在 Result 中。

其次，需要使用静态 newInstance() 方法创建 TransformerFactory。清单 3 展示了所有详细信息：

##### 清单 3. 创建新 TransformerFactory 实例

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

这一步没有太多内容。异常处理与代码本身所用时间一样多。对于 SAX 和 DOM 工厂类，一个异常处理已请求的但不受支持的特性，另一个异常处理实例化错误。

## 恒等转换

TransformerFactory.newTransformer() 的一个版本不接受任何参数（因此无 XSL 样式表）。这允许您执行**恒等转换**，它简单地将输入 XML 从一种形式（比如流）转换为另一种形式（比如 DOM 树）。您以一种格式提供 XML 作为 Source，然后以另一种格式将其推出作为 Result。应该记住这个有用的技巧。

工厂类本身用于获得 Transformer 的实例（在 [下一小节](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#transfrm) 中讨论），并执行简单配置。可以使用 setFeature(String feature, boolean value) 方法来调用处理器上的特性。当然，工厂上设置的任何特性都应用于由此工厂创建的所有 Transformer 实例。

### 创建 Transformer

下一步是获得对象来执行实际转换。这是另一段相当令人厌烦的代码：只在工厂上调用newTransformer()，并为该方法提供要使用的 XSL 样式表。清单 4 展示了详细操作：

##### 清单 4. 使用 TransformerFactory 创建 Transformer

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

**// Get a transformer for this XSL**

**Transformer transformer = factory.newTransformer(inputXSL);**

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

此处没有太多值得注意的地方；惟一需要确保具有的就是 Transformer 与特定样式表之间的连接。因为样式表用于创建 Transformer，这是惟一可以用于该实例的 XSL。如果想要使用不同的样式表执行附加转换，可以重用 TransformerFactory，但必须创建不同的Transformer 实例，以与新样式表连接。

### 执行转换

一切就绪之后，只需要一行代码来执行转换。清单 5 展示了如何使用 transform() 方法。只需为它提供输入 XML 和输出接收器；样式表已经连接到要使用的 Transformer 实例上：

##### 清单 5. 使用 transform() 方法

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

// Get a transformer for this XSL

Transformer transformer = factory.newTransformer(inputXSL);

**// Perform the transformation**

**transformer.transform(inputXML, outputXHTML);**

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

调用该方法之后，转换的结果写出到所提供的 Result 中。在 [清单 5](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code5) 中，这是一个文件，但还可以将输出发送到 SAX ContentHandler 或 DOM Node 中。如果想要全部尝试，绑定的文件提供了简单的 XML 文件、XSL 样式表和源代码（参阅 [下载](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#download)）。

[**回页首**](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 高速缓存 XSL 样式表

这一切都很简单，但这样使用 JAXP 有两个明显的限制：

* 每次执行 transform()，Transformer 对象都要处理 XSL 样式表。
* Transformer 的实例不是线程安全的。在多个线程之间不能使用相同的实例。

这两个限制源自同一问题：Transformer 每次执行转换时，都必须重新处理 XSL。如果该处理出现在多个线程中，就可能发生严重问题。在线程问题之上，必须一再地付出处理 XSL 样式表的成本。毫无疑问，您非常希望知道如何解决这些问题，那就继续读下去。

### 加载模板

尚未讨论过的接口 javax.xml.transform.Templates 与 javax.xml.transform.Transformer 相近。Templates 接口是线程安全的（解决了第二个限制），并代表**已编译**的样式表（解决第一个限制）。在讨论相关概念之前，请查看清单 6：

##### 清单 6. 使用 JAXP Templates 接口

try {

// Set up input documents

Source inputXML = new StreamSource(

new File("phonebook.xml"));

Source inputXSL = new StreamSource(

new File("phonebook.xsl"));

// Set up output sink

Result outputXHTML = new StreamResult(

new File("output-templates.html"));

// Setup a factory for transforms

TransformerFactory factory = TransformerFactory.newInstance();

**// Pre-compile instructions**

**Templates templates = factory.newTemplates(inputXSL);**

**// Get a transformer for this XSL**

**Transformer transformer = templates.newTransformer();**

// Perform the transformation

transformer.transform(inputXML, outputXHTML);

} catch (TransformerConfigurationException e) {

System.out.println("The underlying XSL processor " +

"does not support the requested features.");

} catch (TransformerException e) {

System.out.println("Error occurred obtaining " +

"XSL processor.");

}

[清单 6](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code6) 中的黑体行表示需要从 [清单 5](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#code5) 进行的惟一更改。不是使用工厂来直接获得 Transformer，而是使用 newTemplates() 方法；这将返回一个 Templates 对象，它是线程安全的。可以将该对象传递到其他线程中的其他方法，而且根本无需担心。因为它将从传递它的 XSL 中预编译转换指令，所以传递给其他方法甚至线程是安全的。

然后，从 Templates.newTransformer() 方法中获得 Transformer 实例。在这个阶段无需指定 XSL，因为 Transformer 已经处理过了（事实上，它是已编译的 XSL，所以可以不更改样式表，如果您愿意的话）。除了多出的一行以及对现有行的一个更改之外，再无其他新内容。相当酷，考虑一下您的代码因为这个小小的更改变得有多好。

### 从 Transformer 到 Templates

最后一个值得考虑的问题是，何时使用从工厂直接获得的 Transformer 和何时使用 Templates 对象。我几乎总是选择使用 Templates 对象，因为使用 XSL 时，我总是重复使用相同的样式表。与其花时间在 XSL 上进行多次传递，我宁愿将指令预编译到 Templates 对象中，完成 XSL 处理。

也就是说，在一些情况下，最好是直接从 TransformerFactory 中拖出 Transformer。如果您知道您将只使用特定样式表执行单个转换，那么不预编译到 Templates 对象中会比较快，因为预编译需要略多一点的开销。但是，需要确保没有重用。在我的（完全不科学，使用了简短的样例）测试中，我发现如果使用一个 XSL 样式表两次，使用 Templates 对象和直接使用 Transformer 不分上下。 一旦使用三次以上，Templates 方法就要好多了。您还需要确保将没有任何线程技术问题；但这是一件简单的事情，所以我把它留给您应用在编程中。 一般地，使用 Templates 对象通常更安全。

[**回页首**](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 更改 XSL 处理器

在 [第 1 部分](http://www.ibm.com/developerworks/xml/library/x-jaxp/?S_TACT=105AGX52&S_CMP=cn-a-x) 中已经看到，通过更改系统属性，可以用自己的实现替换默认 JAXP 解析器实现。同一原则适用于 XSL 处理器。JAXP 预包装有 Xalan-J（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)），我总是使用它。但灵活性总是好事，而 JAXP 提供了这一点。

如果想要使用除 Xalan 之外的处理器，请为名为 javax.xml.transform.TransformerFactory 的系统属性提供一个值。需要为该属性分配要实例化的类的名称。该类应继承 javax.xml.transform.TransformerFactory（当然，这也是要设置的系统属性的名称），并填充方法其余的抽象。仅使用如下代码：

java -Djavax.xml.transform.TransformerFactory

=[transformer.impl.class] TestTransformations

simple.xml simple.xsl

全部结束！

[**回页首**](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#ibm-pcon)

## 结束语

在早期版本中，JAXP 不过是位于 SAX 和 DOM 以及那些 API 过时版本之上的小薄层。现在，使用 JAXP 1.3 可以解析、验证并转换 XML，而无需编写一行特定于开发商的代码。虽然追溯到 SAX 代码、使用类似 DTDParser（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp2/#resources)）的工具或甚至自己处理转换，这些都是有意义的，但在 API 和工具的仓库中您需要 JAXP。甚至可能比开发商中立性更重要的一点是，具有最新的 Java 虚拟机 (JVM) 的所有客户和客户机都将具有 JAXP。所以使用它，好好使用它，经常使用它。