**JAXP 全面介绍(第 1 部分)**

**XML 处理工具包使解析和验证变得更容易**

Java API for XML Processing (JAXP) 允许使用几种不同的 API 来验证、解析和转换 XML。JAXP 既提供了使用方便性，又提供了开发商中立性。 本系列介绍 JAXP，由两部分组成。本文是第一部分，向您展示如何利用 API 的解析和验证特性。第二部分介绍使用 JAXP 进行 XSL 转换。

Java 技术和 XML 无疑是最近五年来最重要的编程开发工具。因此，用于在 Java 语言中处理 XML 的 API 就发展起来了。两个最流行的 —— 文档对象模型 (DOM) 和 Simple API for XML (SAX) —— 已经产生巨大的影响，JDOM 和数据绑定 API 也随之产生了。彻底理解其中一个或两个 API 是非常必要的；正确使用全部 API 会让您成为权威。但是，越来越多的 Java 开发人员发现他们不再需要广泛了解 SAX 和 DOM —— 这主要是由于 Sun Microsystems 的 JAXP 工具包。Java API for XML Processing (JAXP) 使得 XML 甚至对于 Java 初级开发人员也变得易于掌握，并大大提高了高级开发人员的能力。也就是说，即使使用 JAXP 的高级开发人员对于他们十分依赖的 API 也有误解。

本文假设您已基本了解 SAX 和 DOM。如果您完全不懂 XML 解析，那么可能需要首先阅读在线参考资料中有关 SAX 和 DOM 的信息，或者浏览我的书（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "resources)）。您不需要精通回调或 DOM Node，但必须至少了解是 SAX 和 DOM 在解析 API。本文还有助于基本了解它们之间的差别。如果您掌握了这些基本知识，本文将对您更有帮助。

**JAXP：是 API 还是抽象？**

严格说来，JAXP 是 API，但更准确地说是抽象层。它没有提供解析 XML 的新方法，没有添加到 SAX 或 DOM，也没有为 Java 和 XML 处理提供新功能。（如果您还不相信这一点，那么阅读这篇文章算对了。）但是，JAXP 使得使用 DOM 和 SAX 来处理一些困难任务变得更容易。它还允许以开发商中立的方式处理一些在使用 DOM 和 SAX API 时可能遇到的特定于开发商的任务。

在 Java 平台的早期版本中，JAXP 是核心平台中单独的下载。在 Java 5.0 中，JAXP 已经是 Java 语言的主要产品。如果已经有最新版本的 JDK（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "resources)），您就已经获得了 JAXP。

没有 SAX、DOM 或另一个 XML 解析 API，则无法解析 XML。我曾经看到过许多关于将 SAX、DOM、JDOM 和 dom4j 与 JAXP 进行比较的请求，但作这样的比较是不可能的，因为前面四个 API 与 JAXP 具有完全不同的用途。SAX、DOM、JDOM 和 dom4j 都解析 XML。JAXP 提供了一种到达这些解析器及其所涉及的数据的方法，但并未提供一种解析 XML 文档的新方法。如果您要正确使用 JAXP，则理解此差别是非常必要的。这还很有可能使您远远领先于您的 XML 开发同行。

如果仍有疑问，请确保您具有 JAXP 发行版（参阅 [逐渐晋级](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "bigtime)）。启动 Web 浏览器并加载 JAXP API 文档。导航至位于 javax.xml.parsers软件包中的 API 的解析部分。令人奇怪的是，您将只找到六个类。这个 API 到底怎么回事？所有这些类都位于现有解析器的顶部。其中两个类仅用于错误处理。JAXP 比人们想像的要简单得多。那么为何会有混淆呢？

甚至 JDOM 和 dom4j（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "resources)）与 JAXP 一样都位于其他解析 API 的顶部。但这两个 API 都提供了从 SAX 或 DOM 中访问数据的不同模型，它们在内部使用 SAX（带有一些技巧和修改）来到达它们提供给用户的数据。

**Sun 的 JAXP 和 Sun 的解析器**

许多解析器/API 混淆来自于 Sun 软件包 JAXP 和该 JAXP 默认使用的解析器。在 JAXP 的早期版本中，Sun 包括 JAXP API（带有刚才提到的六个类和一些常用于转换的类）和 一个叫做 Crimson 的解析器。Crimson 是 com.sun.xml 软件包的一部分。在 JAXP 的新版本中 —— 包括在 JDK 中 —— Sun 已经重新包装了 Apache Xerces 解析器（参阅 [参考资料](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "resources)）。在这两种情况下，虽然解析器是 JAXP 发行版的一部分，但不是 JAXP API 的一部分。

可以认为是 JDOM 附带了 Apache Xerces 解析器。该解析器不是 JDOM 的一部分，但由 JDOM 使用，所以包括它是为了确保 JDOM 可以即装即用。同一原则适用于 JAXP，但并未明确公布：JAXP 附带解析器是为了可以立即使用。但是，许多人将 Sun 的解析器中包括的类作为 JAXP API 本身的一部分。例如，新闻组上的常见问题通常是“我如何使用 JAXP 附带的XMLDocument 类？它的作用是什么？”答案有些复杂。

当我第一次在 Java 1.5 中贸然打开源代码时，我惊奇于我所看到的 —— 或者更应该说是我没有 看到的。没有在正常中的软件包 org.apache.xerces 中找到 Xerces，因为 Sun 将 Xerces 类重新分配给com.sun.org.apache.xerces.internal。（我发现这有点不正常，但没有人问我。）在任何情况下，如果您在 JDK 中查找 Xerces，就能找到它。

首先，com.sun.xml.tree.XMLDocument 类不是 JAXP 的一部分。它是 Sun 的 Crimson 解析器的一部分，包装在 JAXP 的早期版本中。所以这个问题从一开始就令人误解。其次，JAXP 的主要用途是在处理解析器时提供开发商独立性。有了 JAXP，您可以用 Sun 的 XML 解析器、Apache 的 Xerces XML 解析器和 Oracle 的 XML 解析器来处理相同的代码。因而使用特定于 Sun 的类会违反使用 JAXP 的要点。是否弄清楚了本主题是如何变得复杂起来的？JAXP 发行版中的 API 和解析器 已经组合在一起，一些开发人员误将解析器中的类和特性作为 API 的一部分，反之亦然。

既然弄清楚了所有的混淆，那么您就可以深入了解一些代码和概念了。

**[回页首](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "ibm-pcon)**

**SAX 入门**

SAX 是事件驱动的 XML 处理方法。它由许多回调组成。例如，startElement() 回调在每次 SAX 解析器遇到元素的起始标记时被调用。characters() 回调为字符数据所调用，然后 endElement() 为元素的结束标记所调用。许多回调用于文档处理、错误和其他词汇结构。您明白了。SAX 程序员实现一个 SAX 接口来定义这些回调。SAX 还提供一个叫做 DefaultHandler 的类（在 org.xml.sax.helpers软件包中）来实现所有这些回调，并提供所有回调方法默认的空实现。（您将看到，这对于下一节 [处理 DOM](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "dom) 中讨论 DOM 是重点。）SAX 开发人员只需要继承该类，然后实现需要插入特定逻辑的方法。所以 SAX 中的关键是提供这些各种回调的代码，然后让解析器在适当的时候触发其中的一个。下面是典型的 SAX 例程：

使用特定开发商的解析器实现来创建 SAXParser 实例。

注册回调实现（例如，通过使用继承 DefaultHandler 的类）。

开始解析并在回调实现启动时停止。

JAXP 的 SAX 组件提供了完成所有这些操作的简单方法。没有 JAXP，SAX 解析器实例要么必须从开发商类（比如org.apache.xerces.parsers.SAXParser）中直接实例化，要么必须使用一个叫做 XMLReaderFactory 的 SAX 帮助类（也在org.xml.sax.helpers 软件包中）。第一种方法的问题很显然：它不是开发商中立的。第二种方法的问题在于，工厂需要使用解析器类的String 名称作为参数（又是 Apache 类 org.apache.xerces.parsers.SAXParser）。可以通过传递不同的解析器类作为 String 而更改解析器。使用该方法，如果更改解析器名称，则不需要更改任何导入语句，但仍需要重新编译类。这显然不是最好的解决方案。如果能够不重新编译类而更改解析器就方便多了。

JAXP 提供了更好的备选方法：它允许将解析器作为 Java 系统特性。当然，从 Sun 中下载发行版时，您能得到使用 Sun 的 Xerces 版本的 JAXP 实现。更改解析器（比如更改为 Oracle 的解析器）需要更改类路径设置，从一个解析器实现移动到另一个解析器实现。但不 需要重新编译代码。这就是 JAXP 的全部魔力 —— 抽象。

稍叉开一下话题。使用巧妙一点的编码，可以使得 SAX 应用程序从系统特性或特性文件中选择要使用的解析器类。但是，JAXP 提供了相同的行为，且无需任何工作，所以大多数人更愿意走 JAXP 路线。

**SAX 解析器工厂一览**

JAXP SAXParserFactory 类是能够轻易更改解析器实现的关键。必须创建该类的新实例（一会将用到它）。新实例创建之后，工厂提供一种方法用于获得具有 SAX 功能的解析器。实际上，JAXP 实现保护着开发商相关的代码，从而使您的代码完全不受污染。工厂还具有一些其他的有用特性。

除了创建 SAX 解析器实例的基本工作之外，工厂还允许设置配置选项。这些选项影响通过工厂获得的所有解析器实例。JAXP 1.3 中两个常用的选项是，用于设置名称空间意识的 setNamespaceAware(boolean awareness) 和用于打开 DTD 验证的 setValidating(boolean validating)。记住，一旦设置了这些选项，它们将影响在方法调用后从工厂获得的所有实例。

设置了工厂之后，调用 newSAXParser() 会返回 JAXP SAXParser 类立即可用的实例。该类包装底层的 SAX 解析器（SAX 类org.xml.sax.XMLReader 的实例）。它还防止您使用解析器类的任何特定于开发商的附加项。（是否记得上文中有关 XmlDocument 类的[讨论](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "disc_xmldoc)？）该类允许启动实际的解析行为。清单 1 显示如何创建、配置和使用 SAX 工厂：

**清单 1. 使用 SAXParserFactory**

import java.io.OutputStreamWriter;

import java.io.Writer;

// JAXP

import javax.xml.parsers.FactoryConfigurationError;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParserFactory;

import javax.xml.parsers.SAXParser;

// SAX

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class TestSAXParsing {

public static void main(String[] args) {

try {

if (args.length != 1) {

System.err.println ("Usage: java TestSAXParsing [filename]");

System.exit (1);

}

// Get SAX Parser Factory

SAXParserFactory factory = SAXParserFactory.newInstance();

// Turn on validation, and turn off namespaces

factory.setValidating(true);

factory.setNamespaceAware(false);

SAXParser parser = factory.newSAXParser();

parser.parse(new File(args[0]), new MyHandler());

} catch (ParserConfigurationException e) {

System.out.println("The underlying parser does not support " +

" the requested features.");

} catch (FactoryConfigurationError e) {

System.out.println("Error occurred obtaining SAX Parser Factory.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

class MyHandler extends DefaultHandler {

// SAX callback implementations from ContentHandler, ErrorHandler, etc.

}

在 [清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "code1) 中，可以看到在使用工厂时出现两个特定于 JAXP 的问题：无法获得或配置 SAX 工厂，及无法配置 SAX 解析器。第一个问由FactoryConfigurationError 表示，通常发生在无法获得 JAXP 实现或系统特性中指定的解析器时。第二个问题由ParserConfigurationException 表示，发生在请求的特性在所使用的解析器中不可用时。两个问题都易于处理，且不应在使用 JAXP 时造成任何困难。事实上，您可能想要编写代码，来尝试设置几个特征并巧妙处理某个特性不可用时的情况。

SAXParser 实例是在获得工厂、关闭名称空间支持并打开验证时获得的；然后解析开始。SAX 解析器的 parse() 方法采用前面提到的 SAXHandlerBase 帮助类的一个实例，自定义处理器类继承自该类。请参阅代码发行版来查看该类的实现的完整 Java 清单（参阅 [下载](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "download)）。还传递File 以进行解析。但是，SAXParser 类不只包含这一个方法。

**使用 SAX 解析器**

一旦具有 SAXParser 类的实例后，您可以做的远远不止于给它传递 File 来解析。由于大型应用程序中组件的通信方式，所以假设对象实例的创建者就是它的用户并不总是安全的。一个组件可能创建 SAXParser 实例，而另一个组件（可能由另一个开发人员编码）可能需要使用相同的实例。因此，JAXP 提供了确定解析器的设置的方法。例如，可以使用 isValidating() 来确定解析器是否将执行验证，使用isNamespaceAware() 来查看解析器是否可以处理 XML 文档中的名称空间。这些方法可以为您提供关于解析器可以做什么的信息，但只带有SAXParser 实例而非 SAXParserFactory 本身的用户无法更改这些特性。您必须在解析器工厂级别完成这一操作。

还有许多方法来请求文档的解析。并非只能接受 File 和 SAX DefaultHandler 实例，SAXParser 的 parse() 方法还可以接受字符串格式的 SAX InputSource、Java InputStream 或 URL，它们全部具有 DefaultHandler 实例。所以仍可以解析包装在各种格式中的文档。

最后，可以获得底层 SAX 解析器（org.xml.sax.XMLReader 的实例），并直接通过 SAXParser 的 getXMLReader() 方法来使用它。一旦获得该底层实例，一般的 SAX 方法都可用。清单 2 显示 JAXP 中的核心类 SAXParser 类在 SAX 解析中的各种用法的例子：

**清单 2. 使用 JAXP SAXParser 类**

// Get a SAX Parser instance

SAXParser saxParser = saxFactory.newSAXParser();

// Find out if validation is supported

boolean isValidating = saxParser.isValidating();

// Find out if namespaces are supported

boolean isNamespaceAware = saxParser.isNamespaceAware();

// Parse, in a variety of ways

// Use a file and a SAX DefaultHandler instance

saxParser.parse(new File(args[0]), myDefaultHandlerInstance);

// Use a SAX InputSource and a SAX DefaultHandler instance

saxParser.parse(mySaxInputSource, myDefaultHandlerInstance);

// Use an InputStream and a SAX DefaultHandler instance

saxParser.parse(myInputStream, myDefaultHandlerInstance);

// Use a URI and a SAX DefaultHandler instance

saxParser.parse("http://www.newInstance.com/xml/doc.xml",

myDefaultHandlerInstance);

// Get the underlying (wrapped) SAX parser

org.xml.sax.XMLReader parser = saxParser.getXMLReader();

// Use the underlying parser

parser.setContentHandler(myContentHandlerInstance);

parser.setErrorHandler(myErrorHandlerInstance);

parser.parse(new org.xml.sax.InputSource(args[0]));

到此为止，已经针对 SAX 谈了许多，但还没有显示任何显著的或惊人的内容。JAXP 的附加功能相当小，尤其是涉及到 SAX 的地方。这个最小功能使得代码更易移植，让其他开发人员用任何 SAX 兼容的 XML 解析器来自由或商业地使用它。好了。使用 SAX 与 JAXP 再无其他内容。如果已经了解了 SAX，您已经成功了大约 98％。您只需要学习两个新类和一对 Java 异常，然后就可以开始行动了。如果从未用过 SAX，从现在开始也足够了。

**处理 DOM**

如果您认为需要休息一下来对付 DOM 的挑战，那么就休息一下吧。使用 DOM 与 JAXP 和使用 JAXP 与 SAX 几乎完全相同，惟一要做的就是更改类名和返回类型，这就足够了。如果理解 SAX 如何工作和 DOM 是什么，那就根本没有问题。

DOM 和 SAX 的主要差别是 API 本身的结构。SAX 由基于事件的回调集组成，而 DOM 具有内存树结构。在 SAX 中，决不会有需要处理的数据结构（除非开发人员手动创建一个）。因此，SAX 没有提供修改 XML 文档的能力。DOM 提供了此功能。org.w3c.dom.Document 类表示 XML 文档，由表示元素、属性和其他 XML 构造的 DOM 节点组成。所以 JAXP 不需要启动 SAX 回调；它只负责从解析中返回 DOM Document对象。

**DOM 解析器工厂一览**

基本了解了 DOM 以及 DOM 和 SAX 之间的差别之后，就不需要了解其他内容了。清单 3 看起来与 [清单 1](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "code1) 中的 SAX 代码十分相似。首先，获得 DocumentBuilderFactory（与清单 1 中获得 SAXParserFactory 的方法一样）。然后，配置工厂来处理验证和名称空间（与 SAX 中的方法一样）。其次，从工厂中检索与 SAXParser 类似的 DocumentBuilder 实例（与 SAX 中的方法一样）。 然后进行解析，得到的 DOMDocument 对象传递给输出 DOM 树的方法：

**清单 3. 使用 DocumentBuilderFactory**

import java.io.File;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.io.Writer;

// JAXP

import javax.xml.parsers.FactoryConfigurationError;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

// DOM

import org.w3c.dom.Document;

import org.w3c.dom.DocumentType;

import org.w3c.dom.NamedNodeMap;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

public class TestDOMParsing {

public static void main(String[] args) {

try {

if (args.length != 1) {

System.err.println ("Usage: java TestDOMParsing " +

"[filename]");

System.exit (1);

}

// Get Document Builder Factory

DocumentBuilderFactory factory =

DocumentBuilderFactory.newInstance();

// Turn on validation, and turn off namespaces

factory.setValidating(true);

factory.setNamespaceAware(false);

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(new File(args[0]));

// Print the document from the DOM tree and

// feed it an initial indentation of nothing

printNode(doc, "");

} catch (ParserConfigurationException e) {

System.out.println("The underlying parser does not " +

"support the requested features.");

} catch (FactoryConfigurationError e) {

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

"Builder Factory.");

} catch (Exception e) {

e.printStackTrace();

}

}

private static void printNode(Node node, String indent) {

// print the DOM tree

}

}

该代码可以出现两个问题（与 JAXP 中的 SAX 一样）：FactoryConfigurationError 和 ParserConfigurationException。每个问题的原因都于 SAX 中的一样。一个问题出现在实现类中（导致 FactoryConfigurationError），另一个问题是提供的解析器不支持请求的特性（导致 ParserConfigurationException）。在这方面，DOM 和 SAX 之间的惟一差别是，在 DOM 中用DocumentBuilderFactory 替代 SAXParserFactory，用 DocumentBuilder 替代 SAXParser。 仅此一点。（可以查看完整的代码清单，其中包括用于输出 DOM 树的方法；请参阅 [下载](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "download)。）

**使用 DOM 解析器**

一旦拥有 DOM 工厂后，就可以获得 DocumentBuilder 实例。可用于 DocumentBuilder 实例的方法与可用于对应 SAX 实例的方法非常相似。主要差别在于 parse() 方法的变种不接受 SAX DefaultHandler 类的实例。而是返回一个表示已解析的 XML 文档的 DOM Document实例。其余的惟一差别就是两个方法提供了类似 SAX 的功能：

setErrorHandler()，执行 SAX ErrorHandler 实现来处理解析时可能出现的问题。

setEntityResolver()，执行 SAX EntityResolver 实现来处理实体解析。

清单 4 显示这些方法的实际例子：

**清单 4. 使用 JAXP DocumentBuilder 类**

// Get a DocumentBuilder instance

DocumentBuilder builder = builderFactory.newDocumentBuilder();

// Find out if validation is supported

boolean isValidating = builder.isValidating();

// Find out if namespaces are supported

boolean isNamespaceAware = builder.isNamespaceAware();

// Set a SAX ErrorHandler

builder.setErrorHandler(myErrorHandlerImpl);

// Set a SAX EntityResolver

builder.setEntityResolver(myEntityResolverImpl);

// Parse, in a variety of ways

// Use a file

Document doc = builder.parse(new File(args[0]));

// Use a SAX InputSource

Document doc = builder.parse(mySaxInputSource);

// Use an InputStream

Document doc = builder.parse(myInputStream, myDefaultHandlerInstance);

// Use a URI

Document doc = builder.parse("http://www.newInstance.com/xml/doc.xml");

如果您在阅读 DOM 这一节时感到一点乏味，那么您并不孤单；我在编写时也感到有些乏味，因为将已经学过的有关 SAX 的知识应用到 DOM 是如此简单。

**执行验证**

在 Java 5.0（和 JAXP 1.3 中），JAXP 引进一种新方法来验证文档。不是仅仅在 SAX 或 DOM 工厂上使用 setValidating() 方法，而是将验证划分到新 javax.xml.validation 软件包中的几个类中。由于篇幅所限，本文没有详细介绍验证的所有细微差别，包括 W3C XML Schema、DTD、RELAX NG 模式，以及其他约束模型，但如果已经具有一些约束，那么使用新验证模型并确保文档与约束相匹配则相当容易。

**冗余并非总是好事**

您不 应该做的一件事是使用 setValidating(true) 和javax.xml.validation 软件包。您将会得到一些严重错误，其中大多数错误难以解决。最好是养成从不调用setValidating()（默认为假）的习惯，而是使用新的 JAXP 验证框架。

首先，将约束模型（可能是某个磁盘上的文件）转化为 JAXP 可以使用的格式。将此文件加载到 Source 实例中。（我将在第 2 部分更详细地介绍 Source；现在只要知道它表示磁盘某个位置的文档，可以是 DOM Document 或其他文档。）然后，使用SchemaFactory.newSchema(Source) 创建 SchemaFactory 并加载模式，这将返回一个新 Schema 对象。最后，使用此 Schema 对象，用 Schema.newValidator() 方法创建一个新 Validator 对象。清单 5 清楚地显示了全部操作：

**清单 5. 使用 JAXP 验证框架**

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(new File(args[0]));

// Handle validation

SchemaFactory constraintFactory =

SchemaFactory.newInstance(XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);

Source constraints = new StreamSource(new File(args[1]));

Schema schema = constraintFactory.newSchema(constraints);

Validator validator = schema.newValidator();

// Validate the DOM tree

try {

validator.validate(new DOMSource(doc));

System.out.println("Document validates fine.");

} catch (org.xml.sax.SAXException e) {

System.out.println("Validation error: " + e.getMessage());

}

一旦掌握之后，这一切十分简单。亲自输入此代码，或查看完整清单（参阅 [下载](http://www.ibm.com/developerworks/cn/xml/x-jaxp/" \l "download)）。

**更改解析器**

更改 JAXP 工厂类使用的解析器很容易。更改解析器实际上意味着更改解析器工厂，因为所有 SAXParser 和 DocumentBuilder 实例都来自这些工厂。工厂确定加载哪个解析器，所以必须更改工厂。要更改 SAXParserFactory 接口的实现，请设置 Java 系统特性javax.xml.parsers.SAXParserFactory。如果未定义此特性，则返回默认实现（不管开发商指定哪个解析器）。同一规则适用于所使用的 DocumentBuilderFactory 实现。在这种情况下，将会查询 javax.xml.parsers.DocumentBuilderFactory 系统特性。

**结束语**

阅读完本文后，您差不多已经了解到 JAXP 的全部范围：

(1).提供到 SAX 的钩子

(2)提供到 DOM 的钩子

(3)允许容易地更改解析器

要理解 JAXP 的解析和验证特性，您需要阅读比较晦涩的资料。要使 JAXP 运行，最困难的部分是更改系统特性、通过工厂而非解析器或构建器来设置验证，和弄清楚 JAXP 不是 什么。JAXP 为两种流行的 Java 和 XML API 提供了有用的插入层 (pluggability layer)。它使得代码具有开发商中立性，并允许进行解析器之间的更改，而无需重新编译解析代码。所以请下载 JAXP 并使用它！第 2 部分将介绍 JAXP 能够如何帮助转换 XML 文档。