Java Web 服务: Metro 简介

**简介：** Metro Web 服务栈为访问和实现 Web 服务提供综合解决方案。它基于 JAXB 2.x 和 JAX-WS 2.x Java 标准的参考实现，并拥有支持 WS-\* SOAP 扩展技术和实际 Web 服务部署的附加组件。

Metro Web 服务栈是由 Sun Microsystems 开发的一个开源工具。它将 JAXB 2.x 数据绑定和 JAX-WS 2.x Web 服务标准的参考实现与其他与 XML 相关的 Java 标准相结合。Metro 还包括一些附加组件，以支持基本 JAX-WS 服务的定义和使用以及 SOAP 消息交换的各种 WS-\* 扩展。

## 关于本系列

Web 服务构成了 Java 技术在企业计算应用中的关键部分。在本系列文章中，XML 和 Web 服务顾问 Dennis Sosnoski 介绍了对于使用 Web 服务的 Java 开发人员来说比较重要的主要框架和技术。通过跟随本系列的学习，您将了解到该领域的最新进展，并且知道如何使用它们来为您的编程项目提供帮助。

Metro 既可以用作独立的 Web 服务栈，也可以用作开源 Glassfish 应用服务器中的集成组件。如果使用 Glassfish，尤其是在拥有配置基本 Web 服务和 WS-\* 扩展的 GUI 工具的开源 NetBeans IDE 上进行开发时，Web 服务配置会变得更加容易。本系列继续关注 Web 服务。以前的文章讨论 Apache Axis2 的独立使用，而不是包含 Axis2 并支持 GUI 工具的 Apache Axis2 应用服务器。本文也一样，只讨论以独立于 IDE 的方式单独使用 Metro。

**Metro 基础与 Axis2**

本系列之前的文章已经深入讨论了 Axis2，所以本文首先讨论 Metro 与 Axis2 之间的相似点和不同点。两者的相似点比较有限，主要围绕于使用 Web 服务开发代码的常见需求。这两个框架都允许要么从已有的 Java 代码开始构建 Web 服务（但是，除非使用 Jibx2Wsdl 之类的单独的工具，否则 Axis2 对这种方法的支持有限），要么从 WSDL Web 服务描述开始，生成使用或实现服务的 Java 代码。这两个框架都是将服务操作建模为方法调用，而将服务端口类型建模为接口。

Metro 与 Axis2 之间的不同点比相似点要鲜明得多。从基础上讲，Metro 是围绕 JAXB 2.x 和 JAX-WS 2.x 设计的，无意支持这两种技术以外的任何替代技术（除了遗留的 JAX-RPC 的使用）。而 Axis2 则被设计为支持无限种技术，尤其是在 XML 数据绑定领域。虽然它包括对 JAXB 2.x 和 JAX-WS 2.x 的支持，但是二者并无特殊地位。（如果说有的话，也仅限于 JAX-WS 某种意义上是 Axis2 中的第二类方案，原因正如 “[Axis2 中的 JAXB 和 JAX-WS](http://www.ibm.com/developerworks/cn/java/j-jws8.html)” 中所述，无法为 JAX-WS 服务配置 WS-Security 或其他特性）。

从结构上讲，两种栈在处理请求和响应时都使用处理器。Axis2 以这种处理器方法为基础实现模块：基本 SOAP 消息交换的可插拔扩展，用于以高度可配置的方式实现 WS-\* 技术。Metro 支持多种实现处理器的 WS-\* 技术，但是这些技术都被集成到 Metro 引擎，而不是作为可分离组件。Metro 使用的集成方法不如 Axis2 模块灵活，但是在配置和使用 WS-\* 扩展方面有一些优势。

在客户端代码如何使用 WSDL 服务定义方面，这两种栈之间也有不同。Axis2 主要将 WSDL 服务定义用于客户端代码生成，从 WSDL 中提取服务定义信息，并生成代码，以在运行时构造匹配的 Axis2 客户端配置（不过也可以在运行时解析 WSDL 定义）。JAX-WS 2.x，当然还有 Metro，在运行时都需要 WSDL 服务定义，以便构建服务定义。在运行时使用 WSDL 会增加启动开销 — 不过仅限于第一次服务调用 — 没有任何明显的优点。

在服务器端也有不同之处。对于常见的 HTTP 传输，Axis2 通常作为单独的 Web 应用程序（一个 WAR 程序），另外还有任意数量的服务被部署到该 Axis2 Web 应用程序（不过也可以将它打包到应用程序 WAR 中）。可以通过 Web 页面上传部署服务，也可以通过直接将 Axis2 服务 AAR 文件拖放到扩展的 Axis2 Web 应用程序的适当目录中来部署服务。通常，在构建时，Axis2 从 WSDL 服务定义生成各个服务的配置信息，然后将其包括在服务 AAR 文件中。标准的 Axis2 Web 应用程序还通过 Web 页面提供多种监视和控制工具。

相反，Metro 要求为每个 Web 服务应用程序构建单独的 WAR 文件，而 Metro 库 JAR 文件要么包括在 WAR 中，要么包含在类中（作为 HTTP 服务器的一部分），WAR 中还有一个 WEB-INF/web.xml 文件，该文件引用服务和 Metro servlet。当单独使用 Metro 时，还需要创建一个 sun-jaxws.xml 配置文件，该文件提供关于服务配置的附加信息。这些配置文件中的信息与实际 Web 服务类中的 JAX-WS 注释相结合，一同为服务配置 Metro。由于 Metro 被设计为以这种嵌入式的方式使用，所以不提供任何直接监视或控制的工具。

Axis2 和 Metro 都提供集成的 HTTP 服务器支持。对于 Metro，这是通过 JAX-WS 特性 javax.xml.ws.Endpoint 类提供的。Axis2 和 Metro/JAX-WS 的集成 HTTP 服务器都适合在测试中使用，或者作为异步响应端口，但是不能容纳生产 Web 服务，对于生产 Web 服务，可以使用支持 Servlet API 的 Java 应用服务器。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-jws9.html#ibm-pcon)

**示例应用程序**

[代码下载](http://www.ibm.com/developerworks/cn/java/j-jws9.html#download) 部分提供了本系列之前文章中使用的简单的书库管理服务的一个版本，该版本经过了修改，以演示 Metro 的使用。和之前的版本一样，WSDL 服务定义一共定义 4 个操作：

* getBook，用于获取国际标准书号（International Standard Book Number，ISBN）标识的特定图书的详细信息
* getBooksByType，用于获取某种类型的所有图书的详细信息
* getTypes，用于发现现有的图书类型
* addBook，用于将新的图书添加到书库

在 “[Axis2 中的 JAXB 和 JAX-WS](http://www.ibm.com/developerworks/cn/java/j-jws8.html)” 中，您看到了这个应用程序在 Axis2 中如何工作：首先使用 JAXB 2.x 数据绑定生成代码，然后进行 JAX-WS 2.x 服务配置。该文章的大部分内容也适用于使用 Metro 的情况。除了服务名称和端点地址不同外，WSDL 是相同的，生成的 JAXB 数据模型是相同的，甚至生成的服务类也是相同的，只是 Java 包和 JAX-WS 注释中使用的服务名称不同。

**客户端的使用**

与使用 Axis2 和 JAX-WS 相比，在使用 Metro 的情况下，示例应用程序的客户端代码是相同的，甚至构建步骤也是相同的。请参阅 “[Axis2 中的 JAXB 和 JAX-WS](http://www.ibm.com/developerworks/cn/java/j-jws8.html)” 了解代码和处理的详细信息。

**服务器端的使用**

与使用 Axis2 和 JAX-WS 相比，在使用 Metro 的情况下，示例应用程序的服务器端代码也是一样的，只是构建步骤有所不同。使用 Axis2 时，是通过创建一个包含服务和数据模型类的 JAR 文件来准备用于部署的服务，然后通过将该 JAR 拖放到 Axis2 服务器安装目录中的 WEB-INF/servicejars 目录中来部署服务。

而当使用 Metro 时，则需要创建一个包含服务和数据模型类的 WAR 文件，然后创建 Metro 库 JAR（不过也可以将 Metro JAR 直接安装到 Web 服务器 — 如果使用 Tomcat，Metro 下载文件中包括一个用于安装 JAR 的 metro-on-tomcat.xml Ant 构建文件，另外在文档中还有说明），以及一对配置文件。WEB-INF/web.xml 文件配置实际的 servlet 处理。清单 1 显示用于示例应用程序的版本：

**清单 1. 示例应用程序 web.xml**

|  |
| --- |
| <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">  <display-name>MetroLibrary</display-name>  <description>Metro Library Service</description>  <listener>  <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener  </listener-class>  </listener>  <servlet>  <servlet-name>MetroLibraryPort</servlet-name>  <display-name>MetroLibraryService</display-name>  <description>Endpoint for Metro Library Service</description>  <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>MetroLibraryPort</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <session-config>  <session-timeout>60</session-timeout>  </session-config>  </web-app> |

如果以前开发过 Java Web 应用程序，那么 [清单 1](http://www.ibm.com/developerworks/cn/java/j-jws9.html#listing1) WEB-INF/web.xml 文件看上去应该很熟悉（至少结构上是这样）。相应的条目告诉 servlet 引擎将 WAR 文件部署到哪里，以便使用com.sun.xml.ws.transport.http.servlet.WSServletContextListener 类作为 servlet 上下文事件的侦听器，另外使用com.sun.xml.ws.transport.http.servlet.WSServlet 类作为实际的 servlet。这两个类是 Sun 的 Metro 栈特有的，要使用 Metro 必须引用这两个类。servlet 被配置为接收所有传入该 Web 应用程序的请求（通过 <url-pattern>/</url-pattern> 条目）。

[清单 1](http://www.ibm.com/developerworks/cn/java/j-jws9.html#listing1) WEB-INF/web.xml 文件本身只是配置 servlet 引擎，让它使用 Metro 提供的侦听器和 servlet。另有一个文件 WEB-INF/sun-jaxws.xml（清单 2所示）用于配置 Metro，使其将 servlet 接收的请求路由到服务实现代码。

**清单 2. 示例应用程序 sun-jaxws.xml**

|  |
| --- |
| <endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">  <endpoint name="MetroLibraryPort"  implementation="com.sosnoski.ws.library.metro.MetroLibraryImpl"  url-pattern="/"  wsdl-location="WEB-INF/wsdl/library.wsdl"/>  </endpoints> |

[清单 2](http://www.ibm.com/developerworks/cn/java/j-jws9.html#listing2) WEB-INF/sun-jaxws.xml 文件非常简单，只有一个端点定义，其中包括端口名、实现类、请求的匹配模式以及 WSDL 文档位置。WSDL 文档位置是端点定义中唯一的可选项。如果在 sun-jaxws.xml 文件中不指定服务端点的 WSDL 文档，Metro 会在运行时自动生成一个 WSDL 文档。

## 捆绑带来的问题

从 Java SE 6 开始，JAXB 2.x 和 JAX-WS 2.x 参考实现运行时（除了供应商扩展）成为标准 Java Runtime Environment（JRE）库的一部分。其本意是鼓励将这些技术作为 Java 标准使用，但是这也带来一个副作用：为了使用这些技术的更新的版本，可能需要更改 JRE 的安装。

示例应用程序下载文件中使用的 build.xml 将 Metro JAR 文件直接复制到服务 WAR 文件。如果使用 Java SE 5 或者 Java SE 6 JDK/JRE，并且应用程序被部署到 Apache Tomcat 6.0.20 Web 服务器，那么在作者的系统上是可行的。使用 Java SE 6 或更高版本时，如果类装载冲突（例如抛出 ClassCastException，或者未发现 com.sun.xml... 类）导致问题，那么可以使用下面的修复方法：

* 确保使用适合您的系统的最新 JRE 版本，因为更新可能包括更高版本的 JAXB 2.x 和 JAX-WS 2.x。
* 使用 java.endorsed.dirs 系统属性从 Metro lib 目录中指定一个包含 webservices-api.jar 文件的目录（仅限于 webservices-api.jar 文件，因为将其他 JAR 包括进来会导致类装载冲突）作为更新的库的源。（Tomcat 6.0.x 支持这种机制，方法是查找一个 JAVA\_ENDORSED\_DIRS环境变量，并使用它作为系统属性值）。
* 如果以上方法仍然失败，那么在 JRE 安装目录下的 lib 目录中创建一个 endorsed 目录（如果该目录不存在的话），然后将 Metro webservices-api.jar 文件复制到该目录。

对于最后两种技巧，不需要将 Metro webservices-api.jar 包括在服务 WAR 文件中，因为它在 Web 服务器的类路径中是直接可用的。

**构建和运行示例代码**

在运行示例代码之前，首先需要下载和安装当前版本的 Metro（代码经过 1.5 版的测试）（见 [参考资料](http://www.ibm.com/developerworks/cn/java/j-jws9.html#resources)）。另外还需要对解压的示例代码下载文件根目录中的 build.properties 文件进行编辑，将 metro-home 属性的值改为 Metro 安装目录的路径。如果要使用不同的系统或端口上的服务器进行测试，那么可能需要更改 host-name 和 host-port。

要使用 Ant build.xml 构建示例应用程序，打开一个控制台，进入下载文件的根目录，输入 ant。这将首先调用 JAX-WS wsimport工具（包括在 Metro 中），然后编译客户端和服务器，最后将服务器代码打包为 WAR。接着可以将生成的 metro-library.war 文件部署到测试服务器，并在在控制台输入 ant run，尝试运行示例客户端。示例客户端运行，经过一系列对服务器的请求，打印出每个请求的简要结果。

不幸的是，Metro 处理不会完全地执行例子。当 Metro 服务器代码将异常转换成 SOAP Fault 消息时，它还（默认地）发送栈跟踪细节。Metro 客户端代码不能识别响应中的 Fault 数据，只是抛出适当的 Fault 对象，而不会填充包含的数据。在例子代码中，这会导致一个 NullPointerException。

为了改变这种令人惊讶的默认行为，需要在服务器 JVM 上设置一个com.sun.xml.ws.fault.SOAPFaultBuilder.disableCaptureStackTrace=false 属性。（是的，正是这样 — 将disableCaptureStackTrace 属性设为 false，以禁止发送栈跟踪）。通常需要在 Java Web 服务器启动时进行这样的属性设置。对于 Tomcat，可以通过定义一个环境变量来完成这项工作：

|  |
| --- |
| CATALINA\_OPTS=  -Dcom.sun.xml.ws.fault.SOAPFaultBuilder.disableCaptureStackTrace=false |

对服务器配置进行了更改并且重新启动服务器后，应该可以完全运行示例程序。

**结束语**

在本文中，您看到了使用 Metro Web 服务栈的基本知识。由于 Metro 使用 JAX-WS 2.x 注释进行配置，因此 “[Axis2 中的 JAXB 和 JAX-WS](http://www.ibm.com/developerworks/cn/java/j-jws8.html)” 中使用的 JAX-WS 2.x 示例应用程序代码在 Metro 中也可以使用。唯一需要关心的是如何打包代码并将其部署到服务器端，Metro 和 Axis2 在这方面有明显的差异。Metro 使用嵌入式方法为每个服务或服务组创建一个 Web 应用程序（不提供控制或监视功能）。而 Axis2 通常使用一个专用的 Web 应用程序作为任意数量的服务的宿主（通过 Web 页面直接提供基本的控制和监视功能）。

除了基本的 Web 服务消息交换外，Metro 还支持 SOAP 扩展，例如 WS-Security。和服务打包问题一样，Metro 和 Axis2 对这方面的相似问题采取不同的方法。在下一篇文章中，您将看到 Metro 如何处理在前面的文章中使用 Axis2 处理的 WS-Security 例子。