Java Web サービス: CXF の紹介

Apache Software Foundation による Axis2 と並ぶもう 1 つの Web サービス・フレームワークについて学ぶ

Dennis Sosnoski
Architecture Consultant and Trainer
Sosnoski Software Solutions, Inc.

2010年 2月 09日

Apache CXF Web サービス・スタックは、JAXB 2.x データ・バインディング (およびそれに代わるその他のデータ・バインディング) および JAX-WS 2.x サービス構成をサポートしています。以前の記事で紹介した JAXB/JAX-WS と同様に、CXF では XML ファイルを使用して JAX-WS の構成情報を拡張します。この記事では、連載「Java Web サービス」の著者である Dennis Sosnoski が、CXF を使ってクライアントおよびサーバーの開発を行う際の基本事項を説明します。

このシリーズの他の記事を見る

CXF は、Axis2 スタックを提供しているグループである Apache Software Foundation が提供しているもう 1 つの Web サービス・フレームワークです。同じ組織から提供されているとは言え、Axis 2 と CXF とでは、Web サービスの構成手法も、配布手法もかなり異なっています。今回の記事では、CXF によって動作する Web サービスでの JAXB 2.x および JAX-WS 2.x の基本的な使い方について説明するとともに、以前の記事で取り上げた他の 2 つの JAXB/JAX-WS スタックである Axis2 および Metro と、CXF との比較を行います。

CXF の基本、および他のスタックとの比較

ユーザー・インターフェースに関しては、CXF は他の 2 つの Web サービス・スタック、Axis2 および Metro と多くの点で共通しています。この 3 つのスタックではいずれも、既存の Java™ コードから Web サービスを作成することも、WSDL Web サービス記述から Web サービスを使用または実装する Java コードを生成することもできます。さらに他の 2 つのスタックと同じく、CXF はサービス操作をメソッド呼び出しとしてモデル化し、サービス・ポート・タイプをインターフェースとしてモデル化します。

この連載について

Web サービスは、エンタープライズ・コンピューティングにおいて Java 技術が担う重大な役割の一部です。この連載では、XML および Web サービスのコンサルタントである Dennis Sosnoski が、Web サービスを使用する Java 開発者にとって重要になる主要なフレームワークと技術について説明します。この連載から、現場での最新の開発情報を入手して、それら

を皆さんのプログラミング・プロジェクトにどのように利用できるかを知っておいてください。

CXF が Axis2 と共通していて、Metro とは異なる点は、CXF ではさまざまなデータ・バインディング技術のなかから使用する技術を選べることです。JAXB 2.x データ・バインディングのサポートに関しては、CXF は Metro と肩を並べるほどであり、Axis2 よりも優れています。なぜなら、CXF では WSDL からコードを生成する際に JAXB のカスタマイズを使用することができるからです (Axis2 では使用できません)。CXF では JAXB 以外のデータ・バインディング手法も使用することができますが、現状のサポートは Axis2 には及びません。具体的に言うと、CXF で WSDL からコードを生成できるのは、JAXB または XMLBeans データ・バイディングを使用している場合に限られます。

CXF で使われている推奨のサービス構成手法 (CXF の用語では、フロントエンド) は、JAX-WS 2.x アノテーションです。通常この手法は、XML 構成ファイルによって補完されます。CXF における JAX-WS アノテーションのサポートは Metro に引けをとりません。そのため、JAX-WS を使用するには、Axis2 よりも遥かに CXF のほうが適しています (「JAXB and JAX-WS in Axis2」で説明されているように、Axis2 には JAX-WS を使用する上で大きな制約がいくつかあります)。他の JAX-WS 実装の場合と同様、CXF でも、実行時にクライアントがサービス WSDL を使用可能であることが要件となります。

他の 2 つのスタックと同様に、CXF は構成可能なコンポーネントからなるリクエストおよびレスポンスの処理フローを使用します。CXF ではこれらのコンポーネントを「ハンドラー」とは呼ばずに「インターセプター」と呼びますが、用語を別とすれば、どちらも同等のコンポーネントです。CXF には Metro と同じく、WS-Security のサポートとその他の拡張技術が基本ダウンロードの一部として完備されています。Metro と異なる点は、CXF の JAR はモジュールとなっていることです。これは、どの技術を使用しているかによって、アプリケーションの一部として組み込む JAR を自由に選択できることを意味します (CXF インストール・ディレクトリーの /lib/WHICH_JARS ファイルを見ると、各種の一般的な使用ケースのそれぞれに必要な特定の JAR がわかります)。このモジュール方式の欠点は、アプリケーションに必要な JAR の数が大量になってしまう場合があることです。その一方、デプロイメントのサイズを抑えられるという利点もあります。

Metro とのもう 1 つの共通点として、CXF では一般に、多数の Web サービスを 1 つのサーバー・インストール環境にデプロイするのではなく (これは、Axis2 で使用している手法です)、開発者が Web サービスごとに個別の WAR ファイルを作成しなければなりません。とは言え、CXF には本番での使用に適した統合 HTTP サービスが Jetty サーバーという形で用意されています。これは、Axis2 や Metro に統合された単純な HTTP サーバー・サポートよりも柔軟で強力な選択肢となります。

サンプル・アプリケーション

コードのダウンロードには、連載の以前の記事で使用した単純なライブラリー管理サービスを、CXF の使用方法を説明するために変更したバージョンが用意されています。以前の記事と同じく、WSDL サービス定義では以下の 4 つの操作を定義しています。

- `getBook`。ISBN (International Standard Book Number) で識別される特定の本についての詳細を取得するために使用します。

- `getBooksByType`。特定のタイプのすべての本についての詳細を取得するために使用します。
- `getTypes`。入手可能な本のタイプを検索するために使用します。
- `addBook`。新しい本をライブラリーに追加するために使用します。

このサンプル・アプリケーションが Axis2 でどのような動作をするかについては「JAXB and JAX-WS in Axis2」で説明しました。さらに、その後の「Metro の紹介」では、Metro の場合にどのような動作をするかも説明しました。以前の記事で説明した内容のほとんどは、CXF を使用する場合にも当てはまります。使用する WSDL は、サービス名とエンドポイント・アドレスを除けば、まったく同じです。生成される JAXB データ・モデルも同じで、さらには生成されるサービス・クラスについても、Java パッケージ、そして JAX-WS アノテーションで使用するサービス名以外に、何も違いはありません。

クライアント・サイドの使用方法

CXF の場合、サンプル・アプリケーションのクライアント・サイドのコードは、Axis2 や Metor で JAX-WS を使用するためのコードと同じで、ビルド・ステップも非常によく似ています。したがって、JAX-WS リファレンス実装の wsimport ツールの代わりに、CXF の wsdl2java ツールを使用すればよいだけです。このコードとその操作方法についての詳細は、「JAXB and JAX-WS in Axis2」を参照してください。

クライアント・コードは同じとは言え、CXF でのクライアントの振る舞いには 1 つの大きな違いがあります。それはデフォルトで CXF が、コンソールに不愉快なほど大量のロギング詳細を出力することです。CXF は Java ロギング機能を使用するため、この出力を回避するには、ロギング・プロパティー・ファイルの設定を `WARNING` または `SEVERE` の情報だけを出力するように変更した上で、システム・プロパティーがこのファイルを指すように設定する必要があります。サンプル・アプリケーションの Ant build.xml は、JVM パラメーターの `<jvmarg value="-Djava.util.logging.config.file=${build-dir}/logging.properties"/>` という行を使用して、この設定変更を行います。

サーバー・サイドの使用方法

CXF でのサンプル・アプリケーションでは、サーバー・サイドのコードも Axis2 や Metro で JAX-WS を使用するためのコードと同じで、ビルド・プロセスは Metro と非常によく似ています。Axis2 では、サービスをデプロイするための準備として、サービスおよびデータ・モデル・クラスが含まれる JAR ファイルを作成し、その JAR を Axis2 サーバー・システムの WEB-INF/servicejars ディレクトリーにドロップすることによってサービスをデプロイします。一方 Metro と CXF の場合は、サービス・クラスとデータ・モデル・クラス、Metro または CXF のライブラリー JAR、そして構成ファイルが対で含まれる (その一方には、2 つのスタックでの名前とは異なる名前を付けます) WAR ファイルを作成する必要があります。実際のサーブレット処理を構成する場所は、WEB-INF/web.xml ファイルです。リスト 1 に、このファイルのサンプル・アプリケーション用のバージョンを記載します。

リスト 1. サンプル・アプリケーションの web.xml

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <display-name>CXFLibrary</display-name>
  <description>CXF Library Service</description>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
```

```
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:META-INF/cxf/cxf.xml
      classpath:META-INF/cxf/cxf-extension-soap.xml
      classpath:META-INF/cxf/cxf-servlet.xml
     </param-value>
  </context-param>
  <servlet>
    <servlet-name>CXFServlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

リスト 1 の WEB-INF/web.xml ファイルは単なる標準的なサーブレット構成ファイルで、Web アプリケーション・サーバー (Tomcat など) に対し、サーブレット・アプリケーションとどのようにインターフェースを取るかを指示しているに過ぎません。細かな設定項目は Metro でのサーブレット構成ファイルと同様ですが、CXF の場合、`<servlet-class>` は CXF コードの一部となっています。また、`<listener-class>` は Spring Framework クラスを参照します (「参考文献」を参照)。Metro サーブレットの例と同じく、CXF サーブレットはサンプル Web アプリケーションへと送られてくるすべてのリクエストを処理するように構成されています (`<url-pattern>/</url-pattern>` エントリー)。

サーブレットが受信したリクエストをサービス実装コードにルーティングし、サービスの WSDL をオンデマンドで提供するように CXF を構成するためのファイルは別にあります。それが、WEB-INF/cxf-servlet.xml です。リスト 2 に、このファイルを記載します。

リスト 2. サンプル・アプリケーションの cxf-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xmlns:soap="http://cxf.apache.org/bindings/soap"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd">

    <jaxws:endpoint
        id="Processor"
        implementor="com.sosnoski.ws.library.cxf.CXFLibraryImpl"
        wsdlLocation="WEB-INF/wsdl/library.wsdl"
        address="/">
    </jaxws:endpoint>
</beans>
```

リスト 2 の WEB-INF/cxf-servlet.xml ファイルは、実装クラスを持つ単一のエンドポイントと、リクエストに突き合わせるパターン、そして WSDL 文書の場所を定義しています。WSDL 文書の場所は、このエンドポイント定義で唯一のオプションです。cxf-servlet.xml file ファイルでサービス・エンドポイントの WSDL 文書を指定しなければ、CXF が実行時に、JAX-WS アノテーションに基づいて自動的に WSDL 文書を生成します。

サンプル・コードのビルドおよび実行

バンドルの問題

Java SE 6 の時点で、JAXB 2.x および JAX-WS 2.x リファレンス実装のランタイム (ベンダー拡張機能を除く) は、標準 JRE (Java Runtime Environment) ライブラリーの一部となっています。その目的は、Java 標準としてのこの2 つの技術の使用を広めることでしたが、残念なことに、これには副次作用が伴います。それは、これらの技術の新しいバージョンを使用するには、お使いの JRE を変更しなければならないことです。

サンプル・アプリケーションのダウンロードで使用している build.xml は、必要な CXF JAR ファイルをサービス WAR ファイルに直接コピーします。コピーされるファイルには Java SE 5 でビルドするための JAXB および JAX-WS JAR が含まれますが、Java SE 6 でビルドする場合には、ビルドはお使いの JVM にある JAXB と JAX-WS のバージョンに依存します。Java SE 6 以降を使用していて、クラス・ロードの競合が JAXB コードまたは JAX-WS コード内で問題を発生させる場合には、お使いの CXF ディストリビューションに、JVM の互換性に関する何らかの注意書きが含まれていないかどうかを調べてください。

サンプル・コードを試すには、最新バージョンの CXF をダウンロードして、システムにインストールする必要があります (「参考文献」を参照)。このサンプル・コードのテストに使ったのは、2.2.5 リリースです。さらに、サンプル・コードをダウンロードして解凍したそのルート・ディレクトリー内にある build.properties ファイルを編集し、`cxf-home` プロパティーの値を CXF のインストール・パスに変更してください。サーバーを異なるシステムまたはポートでテストする予定であれば、`host-name` と `host-port` も変更する必要があります。

提供されている Ant build.xml を使用してサンプル・アプリケーションをビルドするには、コンソールでダウンロード・コードのルート・ディレクトリーを開き、`ant` と入力します。これにより、最初に CXF の `wsdl2java` ツール (CXF ディストリビューションに付属) が起動された後、クライアントとサーバーがコンパイルされ、最後にサーバー・コードが WAR としてパッケージ化されます。生成されたこの cxf-library.war ファイルをテスト・サーバーにデプロイし、コンソールで `ant run` と入力すれば、サンプル・クライアントを実行してみることができます。サンプル・クライアントはサーバーに対して一連のリクエストを行い、それぞれのリクエストごとに簡潔な結果を出力します。また「クライアント・サイドの使用方法」で説明したとおり、ビルド・ファイルの設定を変更し、サンプル・クライアントの実行時に詳細なログが出力されないように CXF ロギング機能を構成します。

CXF における Spring

リスト 2 の cxf-servlet.xml 構成ファイルでは、Spring Framework の Bean 構成が使用されていることに注目してください。ご存知のとおり、Spring はアプリケーションをアセンブルするために使用できる多数のコンポーネント・ライブラリーが組み込まれたオープンソースのアプリケーション・フレームワークです。Spring Framework には当初から IoC (Inversion of Control: 制御の反転) コンテナーが基本機能として備わっています。IoC コンテナーを利用すると、JavaBean スタイルのソフトウェア・コンポーネントのリンクおよび構成を行い、実行時に Java リフレクションを利用して Bean オブジェクトのプロパティーにアクセスすることができます。

Spring の IoC コンテナーは一般に、依存性情報には XML ファイルを使用します。リスト 2 の cxf-servlet.xml ファイルは、この一般的な Spring 構成の一例です。`<beans>` 要素は個々の Bean 構成を包むラッパーに過ぎません。`<jaxws:endpoint>` 要素はラッパーに包まれた Bean の 1 つで、CXF

はこの要素を特定タイプのオブジェクト (`org.apache.cxf.jaxws.EndpointImpl` インスタンス) に関連付けます。

cxf-servlet.xml ファイルには、この単純な例で使用しているオプション以外にも多数のオプション (例えばサービスのメッセージ・フロー構成など) を指定することができます。すべてのオプションについての詳細は、CXF ドキュメントの JAX-WS 構成に関する情報を参照してください (Frontends/JAX-WS に記載されています)。

JAX-WS アノテーションは別として、CXF スタックでは CXF 内部へのメッセージ・フローの編成を含め、すべての構成に Spring が使用されます。ほとんどの場合、これらの構成の詳細は、CXF JAR に直接組み込まれた XML 構成ファイルを使用して自動的に処理されます (これらの構成ファイルがどのように参照されているかについては、リスト 1 の web.xml ファイルに含まれている `contextConfigLocation` パラメーターの値を見てください)。その一方で、独自に作成した構成ファイルを使用して、この共通フローをオーバーライドすることもできれば、このフローに追加することもできます。その方法についてはこの連載では直接取り上げないので、CXF ドキュメントで詳細を調べてください。

CXF に関する今後の記事

この記事では、CXF Web サービス・スタックで JAXB 2.x データ・バインディングと JAX-WS 2.x のアノテーションをベースとした構成を使用する際の基本的な事項について説明しました。以前の記事で Axis2 スタックと Metro スタックで使用した JAXB/JAX-WS コードを CXF でも使用するには、ビルドに若干の変更を加え、異なるデプロイメントおよび構成ファイルを使用するだけのことです。このスタック間の互換性が、JAXB と JAX-WS を使用することによってもたらされる主な利点、つまりスタック間の容易な切り替えを可能にしています。

CXF の機能は今回の単純な例では説明しきれません。そこで、今後の記事でも引き続き、CXF の機能を取り上げていきます。次回の記事では WS-Security の使用方法に目を向け、CXF での実装を Axis2 および Metro と比較します。

ダウンロード

内容	ファイル名	サイズ
Source code for this article	j-jws12.zip	16KB

著者について

Dennis Sosnoski

Dennis Sosnoski は Java ベースの XML および Web サービスを専門とするコンサルタント兼トレーナーです。専門家としてのソフトウェア開発経験は 30 年以上に渡り、この 10 年間はサーバー・サイドの XML 技術や Java 技術に注力しています。オープンソースのJiBX XML Data Binding フレームワークや、それに関連した JiBX/WS Web サービス・フレームワークの開発リーダーを務め、さらに Apache Axis2 Web サービス・フレームワークのコミッターでもあります。彼は JAX-WS 2.0 および JAXB 2.0 仕様のエキスパート・グループの一員でもありました。