Web 服务客户端自动生成器中 Java 反射机制的运用

Implementing Web Services Client Auto – generated Tool Using Java Reflection

林宏基 （福州大学计算机系 350002）
郭明专 （厦门海关 361001）

摘要：利用 Web 服务客户端自动生成器可将 Web 服务快速地投入使用，本文用 Java 的反射机制来分析并解决了该生成器编写过程中存在的多种动态性问题，使其具备了很强的灵活性，同时还配以详细的源码说明。

关键词：Web 服务 Web 服务客户端自动生成器 Java 反射机制

1 Web 服务

当前，Web 服务已经成为热点。它是建立可互操作的分布式应用程序的新平台，缩小了 Web 应用和传统桌面应用之间互联的鸿沟，提供各种服务间交互和通讯的机制，在各系统间实现松散耦合的综合服务，省去了开发人员为编写每个应用程序的烦琐工作，从而可以集中精力挖掘软件独特的商业价值。Web 服务已经成为动态电子商务的核心。

2 Web 服务客户端自动生成器

随着 Web 服务数量的日益增多，要有选择地将其迅速融入电子商务以满足快节奏的应用需求，就需要系统能够自动根据 Web 服务的描述文件生成客户端程序，然后利用这个客户端模块，将 Web 服务快速集成到自身的商务流程中，为其所用。Web 服务客户端自动生成器是通过对 WSDL 文档的解析，取得各元素的内容，实现用 XML 定义的 WSDL 数据结构到 Java 对象的映射，以此动态生成可以访问 Web 服务的一系列客户端程序。另外，个人用户可能只需访问一些简单的 Web 服务，如用户常用的天气预报、飞机航班、交通信息等查询，但不愿每次分别敲入一堆的网址来访问相应网站，这个问题也可通过客户端自动生成器来解决，但需在原基础上添加用户界面及自动生成的客户端模块。

. Web 服务客户端自动生成器主要由三部分组成：Web 服务 Stub（桩）程序的生成、用户界面的生成和对 Stub 的调用（即通过调用 Stub 程序来获取 Web 服务）。本文客户端自动生成器的第一部分 Stub 程序的生成采用 Axis 的 WSDL2Java 模块实现；由于分布在网络上的 Web 服务是千变万化的，对应的 WSDL（Web Services Description Language）文档也随之不同，这要求用户界面能根据 WSDL 内容的变化自动做出相应的调整。

WSDL 文档的结构组织图如图 1。

如图 1 所示，WSDL 文档元素之间往往是一对多的关系，这些不确定性，以及可扩展性的要求，使得按固定格式进行编写的方式不可行。生成器的第三部分——对 Stub 的调用，应能处理若干个未知的东西，包括未知的方法名、未知的方法返回类型、未知的参数类型列表、未知的传输协议等，这些都只有在第二部分由用户提交请求之后才能确定，基于以上的动态要求，本文采用 Java 的反射机制实现。

在生成器的用户界面部分，首先要将服务的 WSDL 文档的各个元素以网页形式有组织地暴露出来，并配以说明，供用户直观地选择感兴趣的操作，并在相应的参数界面中填入若干参数值，提交后剩余的工作就由 Stub 的调用部分完成。其中，输入参数若为复杂类型（非 Java 基类的自定义类型），应将其递归分解为基本的简单类型。而对 Stub 的调用是用户在界面中选定操作方法并提交相应的参数值后进行的，这时可以确定要调用方法名 Operation，并由 WSDL 文档取得对应的端口名 Port（对应着 PortType）、方法的返回类型和

参数类型 Types。请注意，这些取出的信息都是以字符串形式存在的。

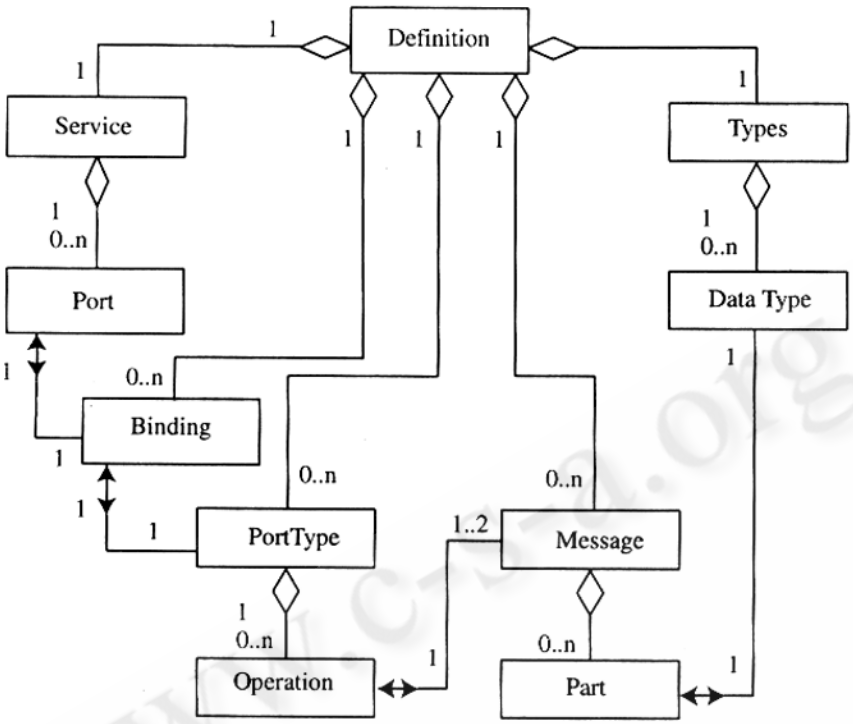

图 1　WSDL 文档的结构组织图

3　Java 反射

反射（reflection）是 Java 编程语言的特性。反射就是一个在当前 JVM 中支持类、接口和对象内省的小型的、类型安全的和可靠的 API。它是通过 java. lang. reflect 包提供的，可以对类的能力进行分析。反射使程序代码能够访问装载到 JVM 中的类的内部信息，主要包括获取已装载类的字段、方法和构造函数的信息，允许编写处理类的代码，这些类是在程序运行时临时确定的，而非源代码中事先选定的。这使反射成为构建灵活的应用的主要工具。

4　使用 Java 反射机制的实现

由于对 Stub 的调用中存在相当多的不确定性，以及功能本身对未来扩展的要求，使 Java 的反射功能在本文得到全面的运用。以下就是在实现上面 Java 程序调用 Stub 步骤中所用到的反射功能：

4.1　确定对象的类型和类型名称

```
Class cls = arguments[i].getClass();
System.out.println("Class name : " + cls.getName());
```

其中，arguments 存放的是方法的参数值对象数组，用于确定当前参数对象的类型。

4.2　根据运行时确定的类型名来创建相应的实例

（1）一般情况，该类型的构造函数无参数

```
Class serviceLocatorClass = Class.forName(ServiceName);
Object serviceLoc = serviceLocatorClass.newInstance();
```

用于创建 Stub 程序的 Locator（根）实例或者复杂类实例。其中，ServiceName 应当是 Service 类的全名，即在前面带有 Java 包名。

（2）对于有参数的情况，采用如下方法

```
Class VarDef = Class.forName(ClassName);
Class params[] = new Class[]{Arg1.Class, Arg2.Class, ...,ArgN.Class};
Constructor con = VarDef.getConstructor(params);
Object variable = con.newInstance(new Object[]{Arg1, Arg2, ...,ArgN});
```

这种情况仅用于已知参数的个数和参数的类型的环境，而对于字符型的单参数特例：

```
Class VarDef = Class.forName(ClassName);
Class params[] = new Class[]{String.class};
Constructor con = VarDef.getConstructor(params);
Object variable = con.newInstance(new Object[]{InializationValue});
```

这种特例正好用于将生成器中用户提交的参数值实例化，因为每个参数值都是以字符型传递过来的。

4.3　使用构造函数的信息

如 4.2（2）所示，对于已知参数的个数和参数的类

型的情况，要先获取对应的构造函数；而 4.2(1) 实际上是 4.2(2) 的特例，它是参数为空(null)的构造器。

4.4 对对象的字段进行取值或者设值

```
Field valueField = arguments[i].getClass().getField("value");
System.out.println("Value :" + valueField.get(arguments[i]));
```

因为，在 WSDL2Java 生成过程中，统一将 InOut 类型的参数(既是输入参数又作为输出参数)转换为 Holder 类型，例如：String 型对应的是 javax.xml.rpc.holders.StringHolder 类型，Double 型对应的是 javax.xml.rpc.holders.DoubleHolder。它们的值都仅有"value"字段用以存放值，这就要求采用上述方法取出 value 字段值来。

4.5 调用特定对象的方法

```
String getPortTypeX = "get" + CapitalizeFirstLetterOfPort;
Method theMethod = serviceLocatorClass.getMethod(getPortTypeX, null);
Object portX = (Object) theMethod.invoke(serviceLoc, null);
```

这是在用户作出选择后，根据所选方法及其所在的 Port 端口，组合成 getter 方法以获取 Stub 中 Port 方法的实例 portX，为后面的具体方法调用做准备，同样，在给复杂数据类型赋值时也会用到 setter 方法。以上是没有输入参数的情况，而在具体方法的调用时则要用到有参数的情况：用户选择的方法不确定导致其参数个数不确定。这种情况下，可采用如下方法：

```
Method theOperMethod = thePortClass.getMethod(operationY, methodParamTypes);
Object returnObj = theOperMethod.invoke(portX, arguments);
```

operationY 是用户选择要调用的方法名，methodParamTypes 是该方法的参数类型数组，arguments 是方法的参数值对象数组，这两个数组都是可动态增长的。

以上程序中，inArgs 为界面中用户提交的数据向量(即某一方法的参数集)，它的大小无法确定，因为每个方法的参数个数都可能不同，所以在创建方法参数列表数组时大小无法确定，但是通过 Java 的反射功能，使用自定义的 growArray() 方法可以实现数组的动态增长。因为数组也是 Java 编程语言中的对象，与其他对象一样，它们都有类。我们能够创建新数组，获得数组对象的长度，读写数组对象的索引值。本生成器中的动态数组除了 arguments(方法的参数值数组)外还包括存放方法的参数类型数组。因为 Java 反射功能中的方法调用指定使用数组，所以此处是无法用 List 或 Map 等类型来替代的。

5 使用 Java 反射机制的一些问题

Java 的反射机制提供了一种动态操作程序成员的多功能途径。它允许程序创建和控制任何类的对象(根据安全性限制)，无需提前硬编码目标类。这些特性使得反射特别适用于创建以普通方式与对象协作的库。例如，反射经常在持续存储对象为数据库、XML 或其它外部格式的框架中使用。

但是反射也有两个缺点。第一个是性能问题。当使用字段和方法的反射功能时要远慢于直接代码。性能问题的程度取决于程序中是如何使用反射的。如果它作为程序运行中相对很少涉及的部分，缓慢的性能将不会是一个问题。仅反射在性能关键的应用的核心逻辑中使用时，性能问题才变得至关重要。对于本文而言，因为 WSDL 文档的一般都很小，所以对性能的影响不大。

对于许多应用程序而言，另一个更严重的缺点在于使用反射会模糊程序中实际要发生的事情。程序员希望能在源代码中直接看出程序的逻辑，而像反射这样绕过了源代码的技术会给日后带来维护问题。反射代码比相应的直接代码更复杂。解决这些问题的最佳方案是保守地使用反射，仅在它可以真正增加灵活性的地方使用。

6 结束语

在 Web 服务客户端自动生成器中，充分利用了 Java 的动态性，主要是关于 Java 的反射机制，包括动态的方法名，动态的类名，动态的参数列表以及动态数组等等。在实际的工作中，单独地使用某项功能可能达不到明显的效果，往往需要将多种功能紧密地结合起来，才能达到最佳。

(下转第 48 页)

(上接第 55 页)

参考文献

1 http://java. sun. com/docs/books/tutorial/reflect/index. html.

2 Axis's home page. http://ws. apache. org/axis.

3 柴晓路,动态安置每个 Web 服务棋子—动态电子商务流程的 Web 服务实现,http://www2. ccw. com. cn/02/0234/b/0234b04_2. asp.

4 谷和启,Web 服务应用 - -Web 系统三层结构的动态电子商务应用, http://tech. 163. com/tm/030524/030524_94990. html.

5 柴晓路、梁宇奇编著,Web Services 技术、架构和应用,电子工业出版社,2003。