基于 CXF 的 SOA 应用设计与实现

Design and Implementation of SOA Application Based on CXF

张同杨 ZHANG Tong-yang
(中国石油勘探开发研究院西北分院计算技术研究所,兰州 730020)
(Institute of Computer Technology of Research, Institute of Petroleum Exploration & Development-Northwest (NWGI), PetroChina, Lanzhou 730020, China)

摘要:本文针对异构系统集成中数据共享问题,通过剖析 Apache CXF 开源框架体系结构及其组件构成,分析 CXF 框架及 SOA 同类框架在不同条件下的性能对比,结合集团级业务信息系统整合特点,给出基于 CXF 框架构建 SOA 的方法。理论分析与系统测试都表明 Apache CXF 在集团级异构信息系统集成方面具有明显优势。

Abstract: Aiming at the problem of data sharing between heterogeneous systems, by analyzing the open source framework architecture of Apache CXF and its components, this paper analyzes the performance comparison between the CXF framework and the other SOA framework under different conditions. Combined with the practice of group business information system integration, this paper gives the method of building SOA application based on CXF framework. Both theoretical analysis and system test show that Apache CXF has the obvious advantages in the integration of heterogeneous information systems.

关键词:面向服务架构;CXF;系统集成;SOA

Key words: service-oriented architecture; CXF; system integration; SOA

中图分类号:TP399 **文献标识码**:A **文章编号**:1006-4311(2017)32-0083-05

DOI:10.14018/j.cnki.cn13-1085/n.2017.32.037

1 SOA

集团级业务系统数据一般涉及总部、区域公司、二级单位、基层单位业务数据,以及诸如身份认证系统、人力资源系统、邮件系统、设备信息系统等其他集团级业务数据,业务数据范围广泛、种类繁多。集团级业务系统持续集成所属单位业务系统、其他集团级业务系统的实时数据,丰富和完善集团级业务系统数据、提高数据质量、降低数据导入导出复杂度、减少人工介入整理数据的工作量,在业务运营实践中具有重要的现实意义。

然而,基于功能或过程的企业架构一般是不同平台、不同开发工具、不同管理规范、不同需求、不同时期研发的多阶段交错复杂业务系统,体系结构、平台技术类型差异较大。这种差异和复杂严重影响了企业业务系统的敏捷性和稳定性,从而影响到企业领导者的业务决策和企业核心能力的发挥。面对瞬息万变的市场需求,企业需要具有快速反应、敏捷生存的能力,从而要求企业管理业务系统必须具有敏捷服务、快速重构、资源重用及自由扩充等特点。

1.1 SOA 设计理念

SOA 设计理念以"支持异质、分散和容错"保持系统灵活性,以编排松散耦合软件服务发布业务逻辑。在企业应用集成领域中,SOA 理念复用企业现有应用,保护企业既有信息投资,对企业中的业务流程进行灵活重构与优化,以服务方式构建业务流程和服务对象,增强业务的适应性和敏捷性。

1.2 SOA 设计原则

SOA 设计保持灵活性和松散耦合,设计原则主要是一致性原则,SOA 架构应采用所有参与者都一致理解的方式整合内容,统一管理,以减少开发、集成及维护工作;服务粒度封装适中原则,即应有利于可用性、可维护性、可操作性、可拓展性及易用性,统筹适度封装操作数量;自治原则,所有服务自我管理,独立部署、版本控制及管理,单一更改不影响整个系统;高内聚原则,一个服务基于一个主题进行合并和组装;松耦合原则,服务之间透明访问,一服务改变不影响其他服务;服务调用原则,平台技术中立,支持多种可传输协议和数据格式。

1.3 SOA 协议栈

SOA 架构通过统一不同厂家和用户的技术标准共识,构建协作规范以屏蔽差异。目前 SOA 技术标准分 3 类:元数据标准、基础通信标准和信息交互标准,从角色功能上划分为传输层、消息层、描述层、管理层、服务组合层和表示层等 6 层,如图 1 所示为 SOA 协议栈层次结构。

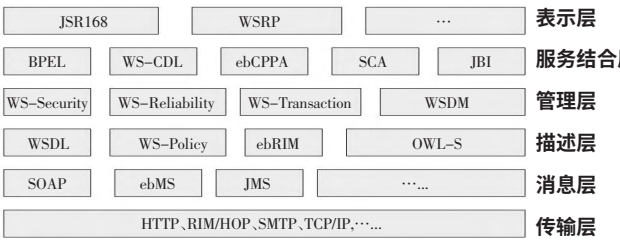

图 1 SOA 协议栈层次结构

2 Apache CXF 基础架构及组件

ApacheCXF 框架是由 ObjectWeb Celti 和 CodeHaus XFire 合并成立。ObjectWeb Celtix 是开源 Java ESB 框架,XFire 则是 SOAP 堆栈。ApacheCXF 融合 ObjectWeb Celtix 和 CodeHaus XFire 两个开源框架的功能,提供实现 SOA 所需要的核心 ESB 功能,包括 SOA 服务创建,服务路由,及 QoS 功能。CXF 支持 JAX-WS 标准,提供对多种 Binding、DataBinding、Transport 以及各种 Format 的支持。CXF 可以根据实际需要,采用代码优先(Code First)或 WSDL 优先(WSDL First)发布和使用 Web Services。Apache CXF 支持 SOAP、XML/HTTP、RESTful HTTP 或 CORBA 等协议,简化 Service 创建,易于和 Spring 无缝集成、易于异构信息系统

基金项目:中国石油天然气集团公司"十二五"规划项目。
作者简介:张同杨(1986-),男,山东菏泽人,工程师,工程硕士,主要研究方向为企业信息化、大数据。

的持续集成，并提供数据安全传输机制。

2.1 Apache CXF 体系结构

CXF 框架不为各组件创建和维护单独的信道，而是使每个组件逻辑上与总线相连，以使所有组件相互连接起来。Bus 组件是 CXF 框架的核心组件，为共享资源提供了一个可配置的场所。CXF 框架使用 Spring 依赖注入、以总线方式串接 Messaging 组件、Interceptors 组件、Transport 组件、Data Binding 组件、Frontend 组件、Service Model 组件、Binding 组件和 WS-Support 组件，以使得各个组件相互通讯。

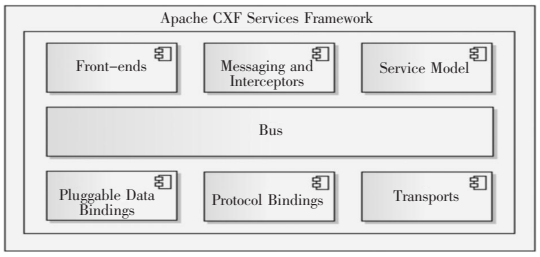

图 2　CXF 体系结构

在具体实现中，抽象类 AbstractBasicInterceptor Provider 实现 InterceptorProvider 接口，CXFBusImpl 继承 AbstractBasicInterptorProvider 类。ExtensionManagerBus 类继承 CXFBusImpl 类以实现 Bus 拓展性。所有对 CXF 的 Data Bindings、Binding、Transport 等功能的引用最终转化为 Bus 组件的方法和内容。图 3 示出了 BUS 组件的实现类结构图。

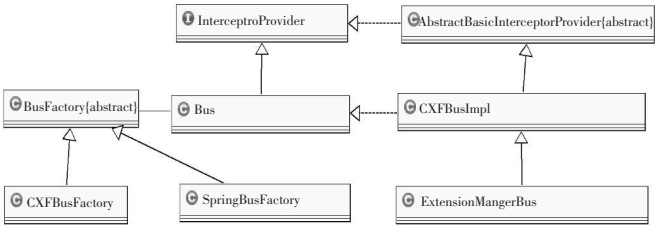

图 3　Bus 组件核心类图

通过对 Bus 进行扩展，可以方便地容纳自己的资源，或替换现有的资源。Bus 默认以 Spring 依赖注入，串接运行态各组件。Bus 默认由 SpringBusFactory 实现，在构造过程中，SpringBusFactory 搜索 META-INF/cxf（就包含在 CXF 的 Jar 中）下的所有 Bean 配置文件，构建单独 ApplicationContext。

2.2 消息组件和拦截器

消息组件、拦截器组件和相位组件共同构成 CXF 框架消息层。Apache CXF 的 Messaging、Phase 和 Interceptors 类结构如图 4 所示。Messaging 接口是 CXF 传递信息的基础，所有操作最终都转化成对一定格式消息的操作。图 4 示出了 CXF 中消息层各组件实现类结构图。

CXF 以消息处理为中心，建立在一个由消息、拦截器和拦截器链（Interceptpr Chain）组成的通用消息层之上。CXF 通过 Interceptors 对消息进行处理，实现诸如消息压缩、SOAP 消息处理等主要功能。拦截器的通用方法主要有 handleMessage 和 handleFault，分别对应消息处理和错误处理。一般在消息传递及处理的过程中，通过拦截器对 CXF 进行安全认证等功能拓展。图 5 示出的代理模式，将在生成目标对象前创建一个具有与原初目标对象相同接口方法的代理，并在真实方法生效前，在代理的方法中添加定制特性方法实现细节业务逻辑。

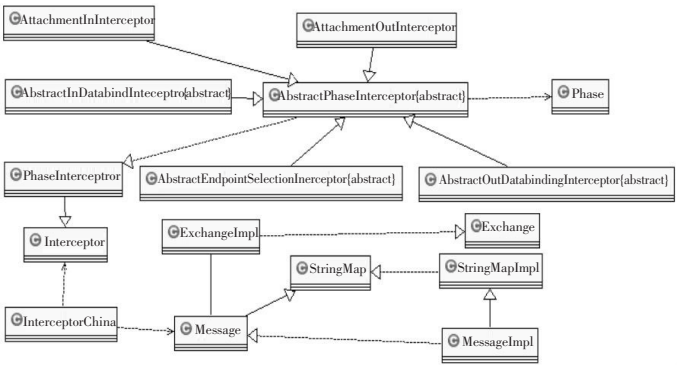

图 4　CXF 的 Messaging、Phase 和 Interceptors 类图

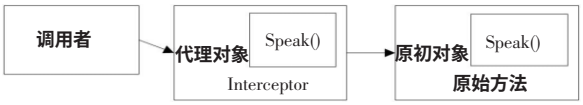

图 5　拦截器代理模式

CXF 框架组合多个拦截器形成拦截器链，其中 InterceptorChain 用于实现增减 Interceptor、控制消息处理及处置异常消息功能。图 6 及图 7 示出 Apache CXF 拦截器客户端实现过程和 Apache CXF 拦截器服务端实现基本过程。

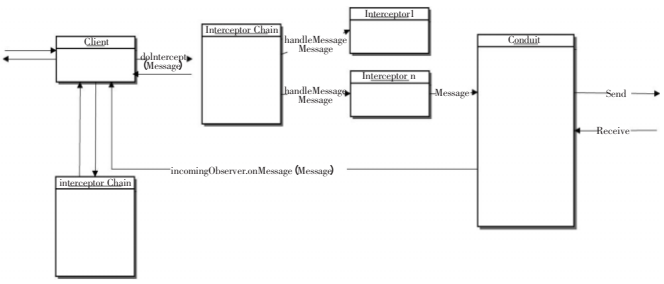

图 6　CXF 拦截器客户端实现过程

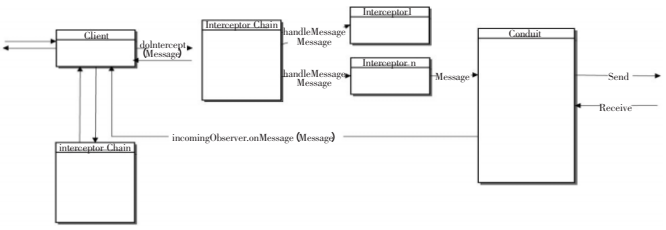

图 7　CXF 拦截器服务端实现过程

2.3 前端编程模型

前端组件（Front-Ends）为 CXF 框架提供了创建服务的编程模型。CXF 服务器端基于 JAX-WS 和 JAX-RS 规范的 Frontend 组件与客户端交互。在实现上，JAX-WS 和 JAX-RS 提供 FactoryBean 部件依赖于 Frontend 组件的 FactoryBean 类，而 FactoryBean 类由依赖于由 Client、Endpoint 及 Server 接口实现的 Endpoint 组件，该组件通过 Interceptor 接口、Fault 接口和 Interceptorchain 接口依赖于 Interceptors 组件。图 8 示出了 CXF 前端应用类结构，ServerFactoryBean 类作用于服务器端点，AbstractServiceFactory 类应用于简化前端模式的实现，而 ClientFactoryBean 类和 ClientProxyFactoryBean 类都基于 ClientFactoryBean 类。（图 8）

2.4 Service 模型

CXF 框架中的服务通过服务模型来表示。它主要有两

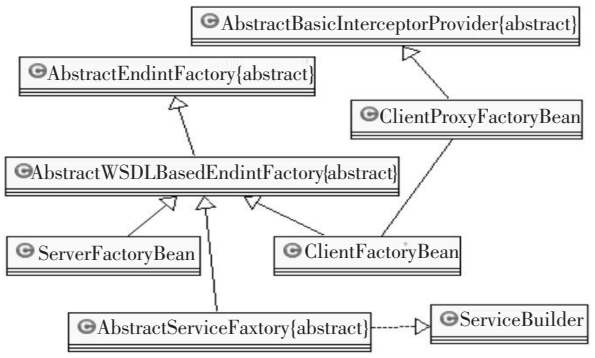

图 8 CXF 前端应用类结构

部分 ServiceInfo 和服务本身。ServiceInfo 作用类似 WSDL，包含接口信息、绑定、端点（EndPoint）等信息；服务则包含了 ServiceInfo、数据绑定、拦截器和服务属性等信息。可使用 Java 类和 WSDL 来创建服务。一般是由前端 ServiceFactory 完成服务的创建。

在具体实现上，动态客户端应用中 Service 组件包括 Service、ServiceBuilder 和 ServiceImpl 三个接口实现 factory、invoker 和 model 组件。其中 model 核心接口是 ServiceInfo 根组件，分别由 InterfaceInfo 组件、BindingInfo 组件和 EndpointInfo 组件继承。

2.5 数据绑定、绑定及传输协议组件

绑定提供了在传输之上映射具体格式和协议的方法，主要的两个类是 Binding 和 BindingFactory。BindingFactory 负责创建 Binding。

Data Bindings 以生成 XML schema 的方式实现 XML 和 Java 之间的映射，数据绑定组件由 Data Bindings 主接口和 Aegis 组件、JAXB 组件、XMLBeans 组件及 MTOM 组件构成。绑定组件由 Bindings 接口（Binding、BindFactory、BindFactoryManager）和 coloc、CORBA、http、object、SOAP 和 XML 六个实现组件构成。

为了对绑定和前端屏蔽传输细节，CXF 框架通过注册 MessageObserver 获得通知的方式，提供用于消息发送的 Conduit 传输对象和 Destination 消息接收对象。在具体实现上，CXF 传输组件由 Transport 接口（Conduits 对象和 Destiations 对象）和 8 个基础业务实现组件（http、http_jetty、http_osgi、https、https_jetty、jms、servlet 和 loca Transport 组件）构成。

3 异构系统集成体系结构

在设计实现中，基于 SOA 设计理念定义的规范和标准，无须了解交互方系统内部实现细节，屏蔽 Windows、Linux 及 Android 等不同平台下的 Java、.NET、C/C++、C# 各类语言的差异，业务系统 CXF 框架作为对外界业务系统交互的 SOA 引擎，通过 webservice 的方式对基于 XML 规范的 WSDL 文件操作，实现异构系统数据共享。图 9 指出了集团级业务系统基于 CXF 框架集成异构业务系统的基本结构。

服务器端调用方法把 XML 格式的 SOAP 请求/响应消息转换成 Java 目标方法调用，客户端调用方法把对 SEI 代理的方法调用转换成 XML 格式的 SOAP 请求和响应，SOAP 消息处理结构如图 10 所示，SOAP 消息的序列化与反序列化。

基于 CXF 基础框架构建 SOA 应用，实现上述序列化

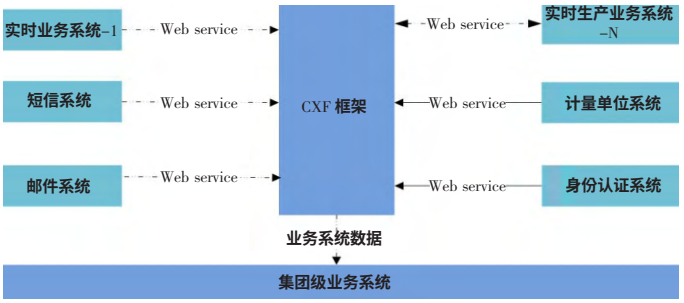

图 9 基于 CXF 框架集成异构业务系统基本结构

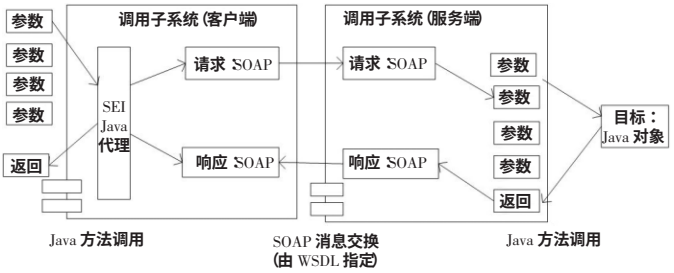

图 10 SOAP 消息的序列化与反序列化

与反序列化过程，一般采用以下基本步骤：

①导入%CXF_HOME%\ lib 下 jar 包配置基础编译环境。

②编写服务接口基类。

```
package com.cnpc.server;
import java.util.List;
import javax.jws.WebParam;
import javax.jws.WebService;
import com.cnpc.pojo.User;
@WebService
public interface HelloWorld {
 String sayHi(@WebParam(name="text")String text);
 String sayHiToUser(User user);
 String[] SayHiToUserList(List<User> userList);
}
```

③编写服务接口的实现类

```
package com.cnpc.server;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import javax.jws.WebParam;
import javax.jws.WebService;
import com.cnpc.pojo.User;
@WebService(
 endpointInterface="com.CNPC.server.HelloWorld",
      serviceName="HelloWorld")
public class HelloWorldImpl implements HelloWorld {
    Map<Integer, User> users =
            new LinkedHashMap<Integer, User>();
    public String sayHi (@WebParam (name = "text")
String text) {
        return "Hello,"+text;
    }
```

```
<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://webservices.cxf.demo/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" targetNamespace="http://webservices.cxf.demo/" name="ProjectManagerImplService":
  - <wsdl:types>
    - <xs:schema xmlns:tns="http://webservices.cxf.demo/" targetNamespace="http://webservices.cxf.demo/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
        <xs:element type="tns:getAllProject" name="getAllProject"/>
        <xs:element type="tns:getAllProjectResponse" name="getAllProjectResponse"/>
        <xs:element type="tns:getDefaultProject" name="getDefaultProject"/>
        <xs:element type="tns:getDefaultProjectResponse" name="getDefaultProjectResponse"/>
        <xs:element type="tns:getProjectName" name="getProjectName"/>
        <xs:element type="tns:getProjectNameByID" name="getProjectNameByID"/>
        <xs:element type="tns:getProjectNameByIDResponse" name="getProjectNameByIDResponse"/>
        <xs:element type="tns:getProjectNameResponse" name="getProjectNameResponse"/>
        <xs:element type="tns:greeting" name="greeting"/>
        <xs:element type="tns:greetingResponse" name="greetingResponse"/>
        <xs:element type="tns:greetingWithParameter" name="greetingWithParameter"/>
        <xs:element type="tns:greetingWithParameterResponse" name="greetingWithParameterResponse"/>
        <xs:element type="tns:registerProject" name="registerProject"/>
        <xs:element type="tns:registerProjectResponse" name="registerProjectResponse"/>
        <xs:element type="tns:removeProject" name="removeProject"/>
        <xs:element type="tns:removeProjectResponse" name="removeProjectResponse"/>
        <xs:element type="tns:setDefaultProject" name="setDefaultProject"/>
        <xs:element type="tns:setDefaultProjectResponse" name="setDefaultProjectResponse"/>
        <xs:element type="tns:updateProjectName" name="updateProjectName"/>
        <xs:element type="tns:updateProjectNameResponse" name="updateProjectNameResponse"/>
      - <xs:complexType name="getProjectNameByID">
        - <xs:sequence>
            <xs:element type="xs:string" name="arg0" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      - <xs:complexType name="getProjectNameByIDResponse">
        - <xs:sequence>
            <xs:element type="xs:string" name="return" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
```

图 11　生成 XML 文档样例

```
public String sayHiToUser(User user) {
        users.put(users.size()+1, user);
        return "Hello,"+user.getName();
}
public String[] SayHiToUserList(List<User> userList) {
        String[] result = new String[userList.size()];
        int i = 0;
        for(User u:userList){
              result[i] = "Hello " + u.getName();
             i++;
        }
        return result;
}
}
```

④服务发布实现类

```
package com.cnpc.server;
import javax.xml.ws.Endpoint;
public class webServiceApp {
/**
 * @param args
 */
public static void main(String[] args) {
        System.out.println("web service start");
        HelloWorldImpl implementor =
                        new HelloWorldImpl();
        String address =
                "http://localhost:8080/helloWorld";
        Endpoint.publish(address, implementor);
        System.out.println("web service started");
}
}
```

⑤验证服务发布情况如图 11。

⑥编写实现客户端实现类

```
package com.cnpc.client;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
import com.cnpc.pojo.User;
import com.cnpo.server.HelloWorld;
public class HelloWorldClient {
/**
 * @param args
 */
public static void main(String[] args) {
        JaxWsProxyFactoryBean jwpfb = new
JaxWsProxyFactoryBean();
        jwpfb.setServiceClass(HelloWorld.class);
        jwpfb.setAddress("http://localhost:8080/helloWorld");
          HelloWorld hw = (HelloWorld) jwpfb.create();
          User user = new User();
          user.setName("某油田");
          user.setDescription("第一采油厂");
          System.out.println(hw.sayHiToUser(user));
        }
}
```

4　传输性能分析

4.1 无附加安全策略测试

图 12 测试时间表明，至少在测试程序中使用的数据，Metro2.0 平均请求处理速度快于 Axis2 及 CXF。在 XML 数据转换中，此三种框架协议栈具有相同的处理速度。这是在 Metro 和 CXF 框架中可以预料的，因为 Metro 和 CXF 都是基于 JAXB 标准实现数据转换的。由此可以判断，Axis2 数据绑定框架实现以及 Axis2 采用也默认采用的数据绑定实现具有与 JAXB 一样的运行速度。

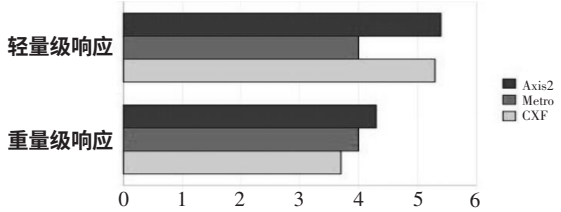

图 12　无附加安全策略测试时间

具有安全策略的测试性能比较，图 13 显示了基于如下安全配置的测试时间：①普通型：无安全策略（数据同图 12）。②用户名令牌：WS 安全策略基于纯文本用户令牌环请求。③标记：WS 安全策略基于消息体和消息头标记时

间戳。④加密标记：安全策略基于标记时间错的消息体和消息头，并加密消息体。图 13，显示了对于 1000 个轻量级请求的指标化测试时间。

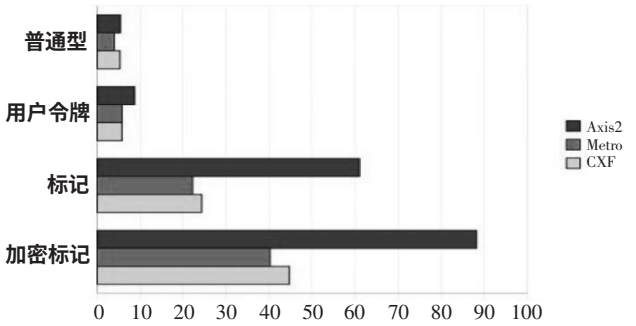

图 13 1000 个轻量级请求指标化耗时对比

4.2 归一化耗时对比

图 14 显示了基于相同的 1000 个请求及轻量级响应的归一化相对时间。以上四种安全配置的测试结果显示，Axis2 总是响应最慢的集成框架。Metro 与 CXF 的差别小于 CXF 与 Axis2:的差别，Metro 框架总是最快的，Metro 框架比 CXF 框架快了 10%，CXF 比 Axis2 快近两倍。图 15 显示基于 100 个重量级响应指标化测试时间对比结果。

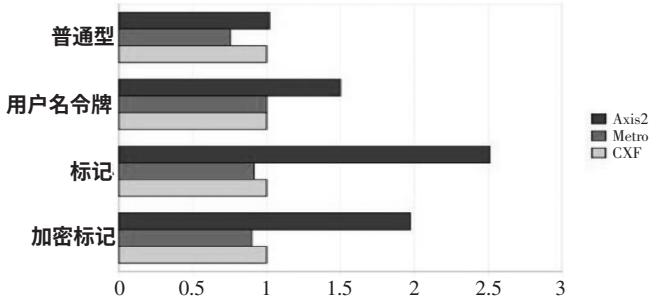

图 14 轻量级响应归一化耗时对比

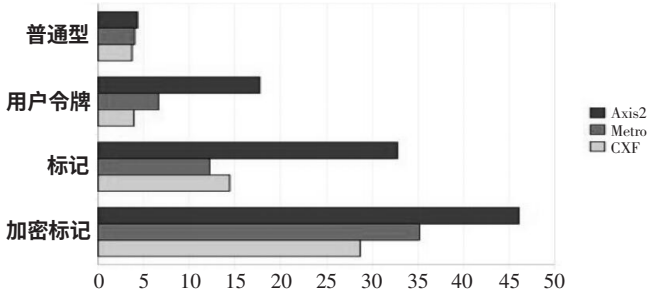

图 15 重量级响应指标化耗时

图 16 显示相对时间（均归一化为 CXF 结果）基于相同的 100 个请求和较重量级响应。在这个测试用例中，Axis2 又一次比 Metro 和 CXF （再次约是 CXF 的一半速度）慢好多，在轻量级消息响应中 Metro 和 CXF 的差别大于逆转 CXF 大约快了 15%。在这个测试用例中，Axis2 又一次比 Metro 和 CXF （再次约是 CXF 的一般速度） 慢好多，在轻量级消息响应中 Metro 和 CXF 的差别大于逆转 CXF 大约快了 15%。基于以上测试报告，Metro 2.0 在基本的请求/响应处理过程中比 Axis 2 及 CXF 2.17 要快。对于 WebService 安全性问题，Metro 的 XWSS 函数库整体上比 Axis2 和 CXF 采用的函数库快。

4.3 性能分析

综合以上测试结果分析，CXF 框架在服务提供应用中对重量级长消息（500KB 及以上）的传输性能比 Axis2 具

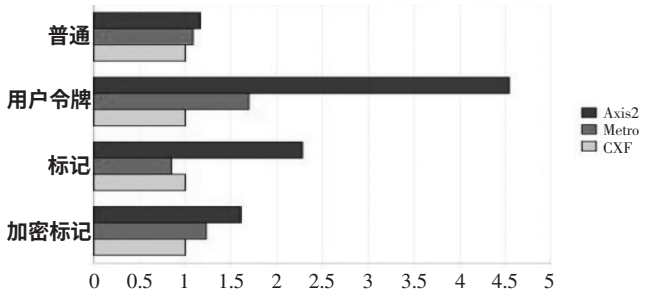

图 16 较重量级响应对比分析

有更短的时间响应时间。

5 结束语

本文在解析 SOA 设计理念的基础上，深入剖析了 Apache CXF 开源框架体系结构及其组件构成，剖析附加安全策略与非附加安全策略的情况下 CXF 传输性能，并给出基于 CXF 异构系统集成的一种实现方法。SOA 和开源框架对各类异构系统的集成在企业业务整合开展过程中具有重要作用。SOA 的实施过程无论采用自顶而下逐步深入细化的过程，还是采用自底向上业务逐步融合的过程，都是一个持续集成生长发展的过程。SOA 实施过程无论从初始、到受控管理、到定义清晰、到量化管理以致持续优化不断成熟并使组织受益，都是一个持续发展、逐步推进的过程。

参考文献：

[1]卢致杰，覃正，韩景倜，王立华，等.SOA 体系设计方法研究[J].工业工程，2004，7（6）：14-16.

[2]汤瀚秋.基于 Axis2 和 CXF 的 Web Service 传输性能测试方案研究[J].软件导刊，2012，11（6）：15-16.

[3]Krafzig D, Banke K, Slama D. Enterprise SOA: Service-Oriented Architecture Best Practices. USA, Prentice Hall PTR, 2004: 41-59.

[4]Nicolai M.著.SOA 实践指南：分布式系统的设计艺术[M].程桦译.北京：电子工业出版社，2008：1-283.

[5]Mark D.Hansen 著.使用 Java Web 服务构建 SOA[M].成保栋译.北京：电子工业出版社，2009：1-535.

[6]SOA Design Patterns [M]. 1st ed. Boston: Pearson Education, Inc, 2009.

[7]A Study Case of Restful Frameworks in Raspberry Pi:A Performance and Energy Overview: Luiz Henrique Nunes, Edvard Oliveira, Julio Cezar Estrella, Stephan Reiff-Marganiec

[8]刘壮业.面向服务架构若干关键问题研究 [J].计算机工程与设计，2009，30（3）：600-603

[9]卢致杰.覃正.韩景倜.王立华.SOA 体系设计方法研究[J].工业工程，2004，7（6）：15-19

[10]Hanscn M.D.著.使用 Java.Web 服务构建 SOA[M].成保栋译.北京：电子工业出版社，2009：1-283.

[11]吴家菊.刘刚.基于面向服务架构的敏捷供应链信息集成研究 [J].计算机工程与设计，2006，27（19）：3445-3448.

[12]Nicolai M.Josuttis 著.SOA 实践指南-分布式系统设计的艺术[M].程桦译.北京：电子工业出版社，2008：1-283.

[13]余浩，朱成，丁鹏，著.SOA 实践-构建基于 Java Web 服务和 BPEL 的企业级应用[M].北京：电子工业出版社，2008：1-310.

[14]Thomas Erl 著.SOA 服务设计原则[M].郭耀译.北京：人民邮电出版社，2009：1-346.

[15]Thomas Erl Clive Gee.Jurgen Kress 著. 下一代 SOA 服务技术与面向服务简明指南[M].卢涛译.北京：电子工业出版社，2015：1-196.