DOI:10.16184/j.cnki.comprg.2012.03.009



# 基于 Spring+CXF 实现用户文件上传的 WebService

孙 磊

**摘 要**：基于 Spring+CXF，实现带有基本验证功能的 WebService，并实现文件上传和相关查询的功能，以便其在使用.NET 或其他框架开发的 MIS 系统中能够自动生成报文并上传。

**关键词**：Spring；WebService；MIS 系统

## 1 引言

Apache CXF 是一个开源的 Services 框架，CXF 帮助您利用 Frontend 编程 API 来构建和开发 Services，像 JAX-WS。这些 Services 可以支持多种协议，比如：SOAP、XML/HTTP、RESTful HTTP 或者 CORBA，并且可以在多种传输协议上运行，比如：HTTP、JMS 或者 JBI，CXF 大大简化了 Services 的创建，同时它继承了 XFire 传统，一样可以天然地和 Spring 进行无缝集成。

## 2 功能分析

（1）该 WebService 需要提供报文上传到服务器上的基本功能。

（2）用户可以查询已经发送的报文历史信息列表，包括报文名称、报文大小和发送时间。

（3）由于不是开放服务，所以还需要对用户身份进行基本认证。

## 3 服务构建

### 3.1 定义服务接口

首先要定义 WebService 输入输出的数据类型，主要包括报文名称、报文类型、报文发送时间以及处理文件流的 DataHandler 属性。代码片段如下：

```
public class MsgItem
{
 //报文名称
   private String filename;
   //报文类型
 private String filetype;
    //传输时间
    private String trans_time;
    //文件流指针
 private DataHandler msgfile;
 public String getFilename()
 {
       return filename;
 }
 public void setFilename(String filename)
 {
       this.filename = filename;
 }
 public String getFiletype()
 {
       return filetype;
 }
 public void setFiletype(String filetype)
 {
       this.filetype = filetype;
 }
 public String getTrans_time()
 {
       return trans_time;
 }
 public void setTrans_time(String trans_time)
 {
       this.trans_time = trans_time;
 }
 public DataHandler getMsgfile()
 {
       return msgfile;
 }
 public void setMsgfile(DataHandler msgfile)
 {
       this.msgfile = msgfile;
 }
}
```

在定义好输入输出的数据结构后，就可以定义要提供的业务服务接口。在该接口中使用 @WebService 注释来声明接口名称和命名空间，使用 @WebResult 注释来声明返回变量名称，使用 @WebParam 注释来声明传入变量名称，代码片段如下：

```
/**
 * 报文上传服务接口
 */
@WebService(name = "MsgTrans", targetNamespace = "
http://soa.test.com/com/test/busw/service/file")
public interface MsgTrans
{
   /**
  * 上传报文
  * @param item 报文实例
  * @return 报文上传结果消息
  */
 @WebResult(name = "up_ret")
 public String UploadMsg
   (
       @WebParam(name = "msg_item")
       MsgItem item
 );
   /**
  * 查询报文信息列表
  * @return 报文信息列表
  */
   @WebResult(name = "msgList")
 public ListObject select_msglist ();
}
```

## 3.2 实现接口

**在完成接口的定义之后，可以开始编写 Java 类来实现上述接口。此处使用 @WebService 指定服务接口的路径。**

**代码片段如下：**

```
@WebService (endpointInterface = "com.test.busw.service.file.MsgTrans")
public class MsgTransServ implements MsgTrans
{
 private String ftphost = "192.168.1.118";
 private String ftpuser = null;
 private String ftpuserpasswd = null;
 private final static String msgoutpath = " msgout/";
 private final static String localpath_err = "err/";
 private static final String MyCharSet = "GB2312";
  // 实现报文上传方法
 @WebResult(name = "up_ret")
 public String UploadMsg (@WebParam (name = "
msg_item") MsgItem item)
 {
      Object obj = SecurityContextHolder.getContext().
getAuthentication().getPrincipal();
(1)----------------------------------------
       if (obj instanceof UserDetails)
       {
            ftpuser = ((UserDetails) obj).getUsername();
  ftpuserpasswd = ((UserDetails) obj).getPassword();
       }
       else
       {
            return "用户验证失败！";
       }
----------------------------------------
       DataHandler handler = item.getMsgfile();
       try
       {
            InputStream is = handler.getInputStream();
            SimpleDateFormat format = new Simple-
DateFormat("yyyyMMddHHmmssSSS");
            String token = format.format(new Date());
            String filename = item.getFilename() + token
+ "." + item.getFiletype();
            OutputStream os = new FileOutputStream
(new File(msgoutpath + filename));
            byte[] b = new byte[100000];
            int bytesread = 0;
            while ((bytesread = is.read(b)) ! = -1)
            {
                 os.write(b, 0, bytesread);
            }
            os.flush();
            os.close();
            is.close();
            sendFtpFiles("in", filename, token);
       }
       catch (Exception e)
       {
            return "上传报文失败:" + e.getMessage();
       }
       return "上传报文成功";
 }

 /**
  * 上传报文至中心服务器
  *
  * @param svrPath
  * @param localPath
  * @throws Exception
  */
 private void sendFtpFiles(String svrPath, String localfile,
String token) throws Exception
 {
       FTPClient ftp = null;
       try
       {
            ftp = new FTPClient();
            ftp.setRemoteHost(this.ftphost);
            ftp.setControlEncoding(MyCharSet);
            ftp.connect();
```

```
            ftp.login(this.ftpuser, this.ftpuserpasswd);
        ftp.setConnectMode(FTPConnectMode.PASV);
                    ftp.setType(FTPTransferType.BINARY);
                    ftp.chdir(svrPath);
                    File file = new File(msgoutpath + localfile);
                    try
                    {
            ftp.put  (msgoutpath + localfile, localfile.replaceAll
(token + ".", "."));
        log(localfile + " sent to ftpserver.(" + file.length() + ")");
                        file.delete();
                    }
                    catch (Exception e)
                    {
                        saveErrFile(localfile);
                        logError("", localfile + " send error." + e);
                        throw e;
                    }
            }
            catch (Exception e)
            {
                    log("FTP 连接错误！ " + e.getMessage());
     throw new Exception("FTP 连接错误！ " + e.getMessage());
            }
            finally
            {
                    try
                    {
                        if (ftp ! = null)
                        {
                            ftp.quit();
                        }
                    }
                    catch (Exception e)
                    {

                    }
            }

    }
    private void logError(String sessionID, String errinfo)
    {
            Logger.getLogger().logToFile(errinfo);
    }
    private void log(String errinfo)
    {
            Logger.getLogger().logToFile(errinfo);
    }
    private void saveErrFile(String filename)
    {
            File file = new File(msgoutpath, filename);
            File errdir = new File(localpath_err, filename);
            file.renameTo(errdir);
    }
    @WebResult(name=" msgList ")
    public ListObject select_msglist ()
    {
          Object obj = SecurityContextHolder.getContext ().
getAuthentication().getPrincipal();
            if (obj instanceof UserDetails)
            {
                ftpuser = ((UserDetails) obj).getUsername();
     ftpuserpasswd = ((UserDetails) obj).getPassword();
            }
            else
            {
                return null;
            }
(2)----------------------------------------
     ApplicationContext ctx = SpringContextUtil.getApplica-
tionContext();
  MsgDao msgdao = (MsgDao) ctx.getBean("MsgDaoProxy");
            List<Object> list = new ArrayList<Object>();
            try
            {
                list = msgdao.select_MsgList(ftpuser);
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
            finally
            {
                msgdao = null;
            }
----------------------------------------
(3)----------------------------------------
              ListObject msglist = new ListObject();
            msglist.setList(list);
            return msglist;
----------------------------------------
    }
   }
```

**这里实现了接口定义的两个方法，一是接收上传的报文，并将接收到的报文以 FTP 方式发送到中心文件服务器上，并给出传输结果；二是报文传输记录查询，实际上是从数据库中查询到用户所发送的报文记录信息，并将结果回传给用户。**

**在该段代码中，有 3 处标号的代码需要做重点说明：**

**（1）处代码的功能就是获取调用方的用户名和密码，并存储到类变量中，在向中心报文服务器上传时，还需要使用此用户名和密码进行 ftp 登录。**Object obj = SecurityContextHolder.

getC ontext（）.getAuthentication（）.getPrincipal（）是利用spring-security组件从Spring容器中获取用户认证信息，只有通过验证的用户名和密码才能够被传递到此处，否则无法调用WebService接口。关于用户认证方面的配置方法，将在后面有详细介绍。

（2）处代码的功能是查询数据库获取报文传输历史记录，此处同样使用spring来获取DAO实例，DAO的接口定义代码如下：

```
public interface MsgDao {
 public List<Object> select_MsgList(String comp);
}
```

按照“面向接口”编程的理念，把数据访问方面所需要的方法都以接口的方式定义好，这样在单元测试、组件化开发、可扩展性等方面会有比较优异的表现。

（3）处代码的功能主要是对List进行封装，因为参照了有关JWS的一个问题中的描述，JDK6.0自带的WebService中WebMethod的参数好像不能是ArrayList或者其他List。传递List需要将List包装在其他对象内部才行，在实践中也遇到了此类问题。通过以下封装的对象即可以传递List对象。

```
@SuppressWarnings("serial")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "listObject", propOrder = { "list" })
public class ListObject
{
 // public void ListObject();
 @XmlElement(nillable = true)
 protected List<Object> list;
 public ListObject()
 {
 }
 public List<Object> getList()
 {
        if (list == null)
        {
               list = new ArrayList<Object>();
        }
        return this.list;
 }
 public void setList(List<Object> list)
 {
        this.list = list;
 }
}
```

到这里为止，完成了向用户提供报文上传服务的基本工作，定义好了WebService接口，并且也实现了将接收到的文件保存到中心服务器上以及从数据库中获取传输记录列表供用户查询。紧接而来的就是如何将这个服务发布为WebService，并且设置只有经过授权的用户才能够使用这些功能呢？

### 3.3 Spring+CXF配置及用户验证的实现

首先，与所有的Web应用程序类似，需要配置web.xml文件来描述Servlet。

配置代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
       "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
 <context-param>
<param-name>contextConfigLocation</param-name>
<param-value>WEB-INF/beans.xml</param-value>
 </context-param>
 <listener>
       <listener-class>
org.springframework.web.context.ContextLoaderListener
       </listener-class>
 </listener>
 <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>
```

以上的配置文件对于做过Servlet开发的人应该是最熟悉的内容，所有的Servlet请求都交给org.apache.cxf.transport.servlet.CXFServlet来处理，并且注册了一个监听器：org.springframework.web.context.ContextLoaderListener，用来加载Spring配置项，Spring配置内容是由contextConfigLocation参数来指定的，即WEB-INF/beans.xml文件作为Spring配置文件。Web.xml的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans xmlns = "http://www.springframework.org/schema/beans"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema -instance" xmlns:aop = "http://www.springframework.org/schema/aop"
  xmlns:tx = "http://www.springframework.org/schema/tx" xmlns:jee="http://www.springframework.org/schema/jee"
  xmlns:jaxws = "http://cxf.apache.org/jaxws" xmlns:sec = "http://www.springframework.org/schema/security"
  xsi:schemaLocation = "http://www.springframework.org/
```

```
schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.0.xsd
http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-2.0.1.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <import resource="spring_lsun.xml" />
    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
    <import resource = "classpath:META -INF/cxf/cxf -servlet.xml" />
    <aop:config>
        <aop:pointcut id="MsgTrans"
    expression="execution(* com.test.busw.service.file.MsgTrans.*(..))" />
  <aop:advisor advice-ref="methodSecurityInterceptor"
                pointcut-ref="MsgTrans" />
    </aop:config>
   <bean id = "methodSecurityInterceptor"class = "org.springframework.security.intercept.method.aopalliance.MethodSecurityInterceptor">
        <property name="validateConfigAttributes">
            <value>false</value>
        </property>
        <property name="authenticationManager">
            <ref bean="authenticationManager" />
        </property>
        <property name="accessDecisionManager">
            <bean class = "org.springframework.security.vote.AffirmativeBased">
                <property name="decisionVoters">
<bean class="org.springframework.security.vote.RoleVoter" />
                </property>
            </bean>
        </property>
        <property name="objectDefinitionSource">
            <ref bean="objectDefinitionSource" />
        </property>
    </bean>
    <bean id = "authenticationManager" class = "org.springframework.security.providers.ProviderManager">
        <property name="providers">
     <beanclass = "org.springframework.security.providers.dao.DaoAuthenticationProvider">
<property name="userDetailsService" ref="userDetailsService" />
            </bean>
        </property>
    </bean>
    <bean id="userDetailsService"
  class = "org.springframework.security.userdetails.memory.InMemoryDaoImpl">
        <property name="userMap">
            <value>
                admin=apmclient,ROLE_ADMIN
                apmclient=apmclientpass,ROLE_USER
        </value>
        </property>
    </bean>
    <bean id = "transAuthen" class = "org.springframework.security.providers.ProviderManager">
        <property name="providers">
     <beanclass = "org.springframework.security.providers.dao.DaoAuthenticationProvider">
                <property name="userDetailsService">
                    <ref bean="UserDAOProxy" />
                </property>
            </bean>
        </property>
    </bean>
   </beans>
```

**此文件从上往下分析，头部的命名空间声明就不多说了，都是按照 cxf 文档来定义就可以了。import 节点定义了需要引入的配置文件名称。aop:config 指定了一个拦截器，表示在执行 com.test.busw.service.file.MsgTrans 类中的所有方法时，都需要执行 methodSecurityInterceptor 所引用的事务。这里如此配置的目的实际上就是为了所有方法在被调用时都需要进行身份验证。methodSecurityInterceptor 以及后面的 authenticationManager、userDetailsService 和 transAuthen 等 bean 的定义，都是根据 spring-scurity 包提供的 acegi 安全框架进行配置。需要注意的是这里用到的 UserDAOProxy 的 bean 定义是在 spring_lsun.xml 文件中定义的，代码片段如下：**

```
    <bean id="lsun_sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocation">
  <value>WEB-INF/ibatis_cnfg/lsun_sqlMapConfig.xml
            </value>
        </property>
        <property name="dataSource">
            <ref bean="servsitedataSource" />
        </property>
    </bean>
    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
```

```
            <property name="dataSource">
                  <ref bean="servsitedataSource" />
            </property>
      </bean>
        <bean id="UserDAO" class="com.test.busw.dbtl.user.
UserDAOImpl">
            <property name="dataSource">
                  <ref bean="servsitedataSource" />
            </property>
            <property name="sqlMapClient">
                  <ref bean="lsun_sqlMapClient" />
            </property>
      </bean>
      <bean id="UserDAOProxy"
            class = "org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
            <property name="transactionManager">
                  <ref bean="transactionManager" />
            </property>
            <property name="target">
                  <ref local="UserDAO" />
            </property>
            <property name="transactionAttributes">
                  <props>
  <prop key="insert*">PROPAGATION_REQUIRED</prop>
  <prop key="update*">PROPAGATION_REQUIRED</prop>
  <prop key="select*">PROPAGATION_REQUIRED,readOnly
                        </prop>
                  </props>
            </property>
      </bean>
```

**这里定义的实际上是** spring+ibatis **实现数据库查询的一个比较常规的配置，**UserDAO **指定了数据库查询实现类，**UserDAOProxy **定义了事务管理代理。**sun_sqlMapClient **指定了** ibatis **的** sqlmap **文件位置和数据源，该数据源是在单独文件中指定的，限于篇幅，不再列出。**

**下面可以先看一下** com.test.busw.dbtl.user.UserDAOImpl **类的代码：**

```
      public class UserDAOImpl extends SqlMapClientDaoSupport implements UserDetailsService
     {
      public UserDetails loadUserByUsername (String username) throws UsernameNotFoundException,
                  DataAccessException
      {
            UserItem user = (UserItem) getSqlMapClientTemplate().queryForObject("User.selectUser",
      username);
            if (user == null)
            {
throw new UsernameNotFoundException("用户名不存在！");
            }
            AuthorityItem role = new AuthorityItem();
            role.setAuthority("ROLE_USER");
      AuthorityItem[] roles = new AuthorityItem[] { role };
            user.setRoles(roles);
            return user;
      }
     }
```

**通过代码可以比较明确地看到该类继承了** SqlMapClientDaoSupport **并实现了** UserDetailsService **接口。**SqlMapClientDaoSupport **是** Spring **包中提供用来与** ibatis **集成的抽象类，这里不再赘述。重要的是** UserDetailsService**，该接口是** Spring-security **包中定义的，与前文所提到的身份验证密切相关。**Spring-security **只是提供一个身份验证框架，具体到每一个项目肯定都会有不同的验证需求，框架通过该接口来降低耦合性，提高框架的稳定性，每个用户在使用此框架进行身份验证时，只需要实现这个接口即可。在上述代码中可以看到，该接口中只有一个方法即：**

```
      public UserDetails loadUserByUsername (String username) throws UsernameNotFoundException,
                  DataAccessException
```

**该方法中，返回类型** UserDetails **也是一个接口。同样是为了保证框架结构的稳定性，用接口来降低耦合。为了满足此方法的要求，定义了一个用户类实现了此接口，代码如下：**

```
      public class UserItem implements UserDetails
      {
       private static final long serialVersionUID = 6714609765732721014L;
       private String username;
       private String password;
       private AuthorityItem[] roles;
       public GrantedAuthority[] getAuthorities()
       {
            return roles;
       }
       public String getPassword()
       {
            return password;
       }
       public String getUsername()
       {
            return username;
       }
       public void setUsername(String username)
       {
            this.username = username;
       }
       public void setPassword(String password)
```

```
    {
        this.password = password;
    }
    public void setRoles(AuthorityItem[] roles)
    {
        this.roles = roles;
    }
    public boolean isAccountNonExpired()
    {
        // 默认永不过期
        return true;
    }
    public boolean isAccountNonLocked()
    {
        // 默认未被锁定
        return true;
    }
    public boolean isCredentialsNonExpired()
    {
        // 默认永不过期
        return true;
    }
    public boolean isEnabled()
    {
        // 默认一直可用
        return true;
    }
}
```

这实际上是一个非常简单的 JavaBean，这里除了 set 方法以外都是实现的接口定义好的方法，换句话说，不管用何种方式，只需要提供框架需要的值就可以了。而实际上返回值类型 GrantedAuthority 也是个接口，因为比较简单，就不再给出实现类 AuthorityItem 的代码了。结合这个 JavaBean，再看之前的 UserDAOImpl 就很好理解了，就是根据用户名从数据库中查询出用户名、密码，构造成 UserItem 对象。根据 Spring-security 框架本身的定义，用户角色也可以是从数据库里查询出来的，而这里因为应用场景比较简单，所以就是直接指定了用户角色为“ROLE_USER”。看到这里，可以再回头看一下 beans.xml 里面定义的 userDetailsService，里面实际上是手工定义了可以接受的角色列表，“ROLE_USER”就是其中一员。

到这里为止，已经完成了全部的服务功能及用户验证要求，最后一步就是完成 WebService 入口配置文件：cxf-servlet.xml。该文件实际上就是指定了所提供服务的一系列属性，配置内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
  <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/beanshttp://www.springframework.org/schema/beans/spring-beans-2.0.xsd
     http://cxf.apache.org/bindings/soaphttp://cxf.apache.org/schemas/configuration/soap.xsd
          http://cxf.apache.org/jaxwshttp://cxf.apache.org/schemas/jaxws.xsd">
    <jaxws:endpoint id="MsgTrans"
        implementor = "com.lygedi.busw.service.file.MsgTransServ" address="/MsgTrans">
        <jaxws:features>
  <bean class="org.apache.cxf.feature.LoggingFeature" />
        </jaxws:features>
        <jaxws:inInterceptors>
  <bean class="springSecurity.SecurityInInterceptor">
    <property name="authenticationManager">
                    <ref bean="transAuthen"/>
                </property>
                </bean>
    <bean class="springSecurity.SecurityOutInterceptor" />
        </jaxws:inInterceptors>
        <jaxws:properties>
          <entry key="mtom-enabled" value="true" />
        </jaxws:properties>
    </jaxws:endpoint>
  </beans>
```

这里明确指定了服务名称及入口程序地址，并且还指定了输入输出安全验证相关配置。最重要的是<entry key=" mtom-enabled" value=" true" />这里的配置，用来实现 WebService 的大文件上传和下载功能。开启基本用户验证和 mtom 模式的 WebService，通过.NET 客户端访问时，需要修改其自动生成的 WebService 引用类，并安装 wse3.0 支持。具体的解决方案可以在网络上进行搜索。

## 4 结语

Java 开源框架的特点就是各负其责，各框架间通过各种接口进行集成。Spring 作为一种 IOC 容器，起到了粘合剂的作用，但同时也带来了大量的配置工作，这就需要我们在开发过程中细心、细心、再细心。否则，可能仅仅由于非常细微的疏忽，而导致整个系统无法运行。

## 参考文献

[1] Rod Johnson，Juergen Hoeller，等. Spring 框架高级编程. 蒋培，译. 机械工业出版社，2006.

[2] 梁爱虎. SOA 思想、技术与系统集成应用详解. 电子工业出版社，2007.

（收稿日期：2011-09-12）