CXF 拦截器 拦截请求和发送时的报文

@author: *donnie*

cxf 可以对请求和发送时的报文进行拦截，然后对其进行加工成我们想要的格式，然后再放到回去！！！！！

我实现的是自定义拦截器

1.在需要拦截的方法上面加上这个注解，注解里面是我们的自定义拦截器

```
@OutInterceptors(interceptors = {
"com.deloitte.tms.vat.webservice.result1.CDATAOutInterceptor" })
@InInterceptors(interceptors="com.deloitte.tms.vat.webservice.result1.ArtifactOu
tInterceptor")
```

2.对返回时的格式进行拦截实例，如果

```
public CDATAOutInterceptor() {
        //这里代表流关闭之前的阶段，这很重要！可以到官网去看，拦截的阶段分为很多种
        super(Phase.PRE_STREAM);
}
```

Phase.PRE_STREAM写的阶段不对可能会对想要的结果有影响，我开始写的Send和Write都是不生效的，网上也有好多博文写的是这种！！！踩到坑了！
复制代码

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import org.apache.cxf.helpers.IOUtils;
import org.apache.cxf.io.CachedOutputStream;
import org.apache.cxf.message.Message;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class CDATAOutInterceptor extends AbstractPhaseInterceptor<Message> {
    private static final Logger log =
LogManager.getLogger(ArtifactOutInterceptor.class);

    public CDATAOutInterceptor() {

        //这里代表流关闭之前的阶段，这很重要！可以到官网去看，拦截的阶段分为很多种
        super(Phase.PRE_STREAM);
    }

    @Override
    public void handleMessage(Message message) {

        /*这里是通过注解的方式来加<!CDATA[[]]>的格式
```

```
         * message.put("disable.outputstream.optimization", Boolean.TRUE);
         * XMLStreamWriter writer =
         * StaxUtils.createXMLStreamWriter(message.getContent(OutputStream.class
         * )); message.setContent(XMLStreamWriter.class, new
         * CDATAXMLStreamWriter(writer));
         */

        try {

            OutputStream os = message.getContent(OutputStream.class);
            CachedStream cs = new CachedStream();

            message.setContent(OutputStream.class, cs);

            message.getInterceptorChain().doIntercept(message);

            CachedOutputStream csnew = (CachedOutputStream)
message.getContent(OutputStream.class);
            InputStream in = csnew.getInputStream();

            String xml = IOUtils.toString(in);
            System.out.println("replaceBegin" + xml);

            //转换的方式
            xml = xml.replace("<body>", "<![CDATA[<?xml version=\"1.0\"
encoding=\"UTF-8\"?><body>")
                    .replace("<returnMessage>", "<returnMessage><![CDATA[")
                    .replace("</returnMessage>", "]]>
</returnMessage>").replace("<returnCode>", "<returnCode><![CDATA[")
                    .replace("</returnCode>", "]]></returnCode>");
            // 这里对xml做处理，处理完后同理，写回流中
            System.out.println("replaceAfter" + xml);
            IOUtils.copy(new ByteArrayInputStream(xml.getBytes()), os);
            cs.close();
            os.flush();
            message.setContent(OutputStream.class, os);

        } catch (Exception e) {
            log.error("Error when split original inputStream. CausedBy : " +
"\n" + e);
        }
    }

    private class CachedStream extends CachedOutputStream {

        public CachedStream() {

            super();

        }

        protected void doFlush() throws IOException {

            currentStream.flush();

        }

        protected void doClose() throws IOException {
```

```
        }

        protected void onWrite() throws IOException {

        }

    }

}
```

复制代码
如果上请求的数据量过大的话：会出现传输的xml获取不到数据的情况。需要将CachedStream换成ByteArrayOutputStream，

如果上请求的数据量过大的话：会出现传输的xml获取不到数据的情况。需要将CachedStream换成ByteArrayOutputStream，

如下的方式：

```
@Override
public void handleMessage(Message message) {
    try {
        OutputStream os = message.getContent(OutputStream.class);
        ByteArrayOutputStream cs=new ByteArrayOutputStream();

        message.setContent(OutputStream.class, cs);

        message.getInterceptorChain().doIntercept(message);

        //ByteArrayOutputStream csnew = (ByteArrayOutputStream)
message.getContent(ByteArrayOutputStream.class);
        //InputStream in = csnew.toString(charsetName);
        //String xml = IOUtils.toString(in);
        String xml=new String(cs.toByteArray(), "utf-8");
        log.info("replaceBegin"+xml);
        xml = xml.replace(""0;
        // 这里对xml做处理，处理完后同理，写回流中
        log.info("replaceAfter"+xml);
        IOUtils.copy(new ByteArrayInputStream(xml.getBytes()), os);
        cs.close();
        os.flush();
        log.info("将参数设置会cxf框架中------------");
        message.setContent(OutputStream.class, os);

    } catch (Exception e) {
        log.error("Error when split original inputStream. CausedBy : " + "\n" +
e);
    }
}
```

3.对请求进行拦截

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.helpers.IOUtils;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
public class ArtifactOutInterceptor extends
AbstractPhaseInterceptor<SoapMessage>{
    private static final Logger log =
LogManager.getLogger(ArtifactOutInterceptor.class);


    public ArtifactOutInterceptor() {
        //这儿使用pre_stream，意思为在流关闭之前
        super(Phase.PRE_STREAM);
    }
    @Override
    public void handleMessage(SoapMessage message) {
        InputStream is = message.getContent(InputStream.class);
        if (is != null) {
            try {
                String str = IOUtils.toString(is);
                log.info("原格式--传入的xml格式为："+str);
                str=str.replace("<", "<");
                str=str.replace(">", ">");
                str=str.replace("<?xml version=\"1.0\" encoding=\"UTF-8\"?
>","");
                InputStream ism = new ByteArrayInputStream(str.getBytes());
                message.setContent(InputStream.class, ism);
                log.info("解析后的格式--传入的xml格式为："+str);
            }
            catch(IOException e){
                log.error("WebService消息拦截器处理异常！",e);
            }
        }

    }

}
```

4.1.对输出加<CADATA[[]]>的另外一种方法

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.xml.stream.XMLStreamWriter;

import org.apache.cxf.helpers.IOUtils;
import org.apache.cxf.io.CachedOutputStream;
import org.apache.cxf.message.Message;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.cxf.staxutils.StaxUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

```
public class CDATAOutInterceptor extends AbstractPhaseInterceptor<Message> {
   private static final Logger log =
LogManager.getLogger(ArtifactOutInterceptor.class);

   public CDATAOutInterceptor() {
      super(Phase.PRE_STREAM);
   }

   @Override
   public void handleMessage(Message message) {

      message.put("disable.outputstream.optimization", Boolean.TRUE);
      XMLStreamWriter writer =
      StaxUtils.createXMLStreamWriter(message.getContent(OutputStream.class
      )); message.setContent(XMLStreamWriter.class, new
      CDATAXMLStreamWriter(writer));

   }

}
```

```
import java.util.Arrays;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

import org.apache.cxf.staxutils.DelegatingXMLStreamWriter;

public class CDATAXMLStreamWriter extends DelegatingXMLStreamWriter{
    private String currentElementName;
    private static String[] CDATA_ELEMENTS = { "returnCode", "returnMessage",
            "fpdm", "fphm","code",
            "smrq","kpfsh","kprq","fpje","fpse", "sl","spfsh",
"spfmc","fply","fplx", "xgrq","zfbz"
            , "rzrq","rzzt","body","start_time","end_time"};
    public CDATAXMLStreamWriter(XMLStreamWriter writer) {
        super(writer);
    }

    @Override
    public void writeCharacters(String text) throws XMLStreamException {
        boolean useCData = isNeedCData();
        if (useCData) {
            super.writeCData(text);
        } else {
            super.writeCharacters(text);
        }
    }

    private boolean isNeedCData() {
        if(currentElementName.equals("body")||currentElementName.equals("data"))
{
            System.out.println("-------");
        }
        if (Arrays.asList(CDATA_ELEMENTS).contains(currentElementName)) {
            return true;
```

```
        } else {
            return false;
        }

    }

    public void writeStartElement(String prefix, String local, String uri)
            throws XMLStreamException {
        currentElementName = local;
        super.writeStartElement(prefix, local, uri);
    }

}
```

二、示例：CXF拦截器处理特殊字符——可配置化

- 配置表

```
SELECT T.*,ROWID FROM MTS_SYS_PARAM T WHERE T.CODE IN ('FPS_WEBSERVICE_SPECIAL','FPS_WEBSERVICE_SPECIAL_CHARACTERS');
```

	GID	CODE	NAME	VAL	SUSER	SCOPE	REMARK
1	c59b5848e2dd4d7083e0ff9e85870961	FPS_WEBSERVICE_SPECIAL	FPS接口特殊字符处理配置	erpWebService		SYSTEM	FPS接口特殊字符处理配
2	52ed2fdd6262436696b1cf564a7ae701	FPS_WEBSERVICE_SPECIAL_CHARACTERS	FPS接口特殊字符处理配置	especialRequest		SYSTEM	FPS接口特殊字符处理配

- 拦截器配置文件：unimax-a-webservice.xml

```
<?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:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa.xsd"
    default-lazy-init="true">
    <!-- webservice -->
    <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" />

    <!-- 输入日志拦截器 -->
```

```
    <bean id="xmlPreHandleInterceptor"
class="com.epichust.project.utils.XmlPreHandleInterceptor" />

    <jaxws:endpoint id="erpWebService"
implementor="com.epichust.fps.project.webService.server.impl.ErpWebService"
address="/erpWebService">
        <!-- 输入日志拦截器 -->
        <jaxws:inInterceptors>
            <ref bean="xmlPreHandleInterceptor"/>
        </jaxws:inInterceptors>
    </jaxws:endpoint>


</beans>
```

- 拦截器定义：XmlPreHandleInterceptor.java

```
package com.epichust.project.utils;

import cn.hutool.core.util.StrUtil;
import com.epichust.mestar.logging.MestarLogger;
import com.epichust.mestar.management.manager.ISysParamManager;
import com.epichust.mestar.utils.spring.SpringContextHolder;
import org.apache.commons.lang3.StringUtils;
import org.apache.cxf.binding.soap.SoapMessage;
import org.apache.cxf.helpers.IOUtils;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.io.CachedOutputStream;
import org.apache.cxf.message.Exchange;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.cxf.service.Service;
import org.apache.cxf.service.invoker.MethodDispatcher;
import org.apache.cxf.service.model.BindingOperationInfo;

import java.io.*;
import java.lang.reflect.Method;
public class XmlPreHandleInterceptor extends
AbstractPhaseInterceptor<SoapMessage>
{
    public XmlPreHandleInterceptor() {
        //这儿使用pre_stream，意思为在流关闭之前
        super(Phase.PRE_STREAM);
    }

    public void handleMessage(SoapMessage message) throws Fault
    {
        Boolean hasSrv = false;
        String basePath = (String) message.get(SoapMessage.BASE_PATH);
        ISysParamManager sysParamManager =
SpringContextHolder.getBean("sysParamManager");
        String svrList =
sysParamManager.getSysParamValByCode(ApsProjectConstant.FPS_WEBSERVICE_SPECIAL);
        if(StringUtils.isNotEmpty(svrList)){
            for (String svrName : svrList.split(StrUtil.COMMA))
            {
```

```
 if(svrName.equals(StringUtils.substringAfterLast(basePath,"/ws/"))){
                    hasSrv = true;
                    break;
                }
            }

        if(hasSrv){
            InputStream is = message.getContent(InputStream.class);
            try
            {
                if (is != null)
                {
                    String str = IOUtils.toString(is);
                    MestarLogger.debug("原格式--传入的xml格式为：" + str);
                    String splArrStr =
sysParamManager.getSysParamValByCode(ApsProjectConstant.FPS_WEBSERVICE_SPECIAL_C
HARACTERS);
                    if(StringUtils.isNotEmpty(splArrStr)){
                        for(String splChar:splArrStr.split(StrUtil.COMMA)){
                            String xmlColumnPre =
StrUtil.addSuffixIfNot(StrUtil.addPrefixIfNot(splChar,"<"),">");
                            //转换的方式
                            str = str.replace(xmlColumnPre,
StrUtil.addSuffixIfNot(xmlColumnPre,"<![CDATA[");
                            String xmlColumnSub =
StrUtil.addSuffixIfNot(StrUtil.addPrefixIfNot(splChar,"</"),">");
                            str = str.replace(xmlColumnSub,
StrUtil.addPrefixIfNot(xmlColumnSub,"]]>"));
                        }
                    }
                    MestarLogger.debug("处理后格式--传入的xml格式为：" + str);
                    InputStream ism = new
ByteArrayInputStream(str.getBytes("UTF-8"));
                    message.setContent(InputStream.class, ism);
                }
            } catch (IOException e)
            {
                MestarLogger.error("WebService消息拦截器处理异常！", e);
            }
            finally
            {
                try
                {
                    if (is != null)
                    {
                        is.close();
                    }
                } catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

- 静态常量定义：ApsProjectConstant.java

```
package com.epichust.project.utils;

public class ApsProjectConstant
{
    /**接口特殊字符处理*/
    public static final String FPS_WEBSERVICE_SPECIAL =
"FPS_WEBSERVICE_SPECIAL";
    public static final String FPS_WEBSERVICE_SPECIAL_CHARACTERS =
"FPS_WEBSERVICE_SPECIAL_CHARACTERS";
}
```