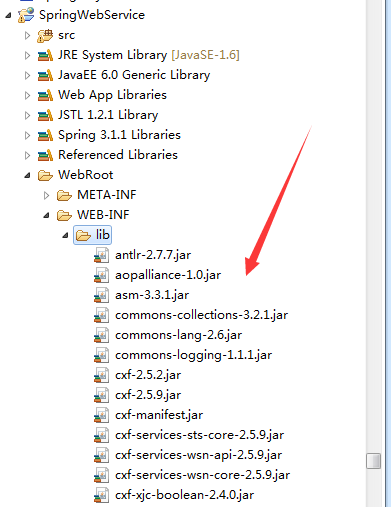
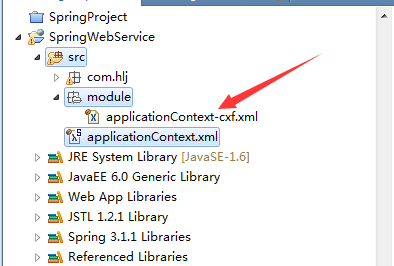
# 1、先建立一个spring项目，再引入cxf 的jar包



# 2、spring配置文件中添加一个关于cxf的配置文件（其实可以写到一起，但是我怕有点乱，就分开写了，这样比较规范）

## 1、spring的配置文件 ApplicationContext.xml 添加cxf的配置文件



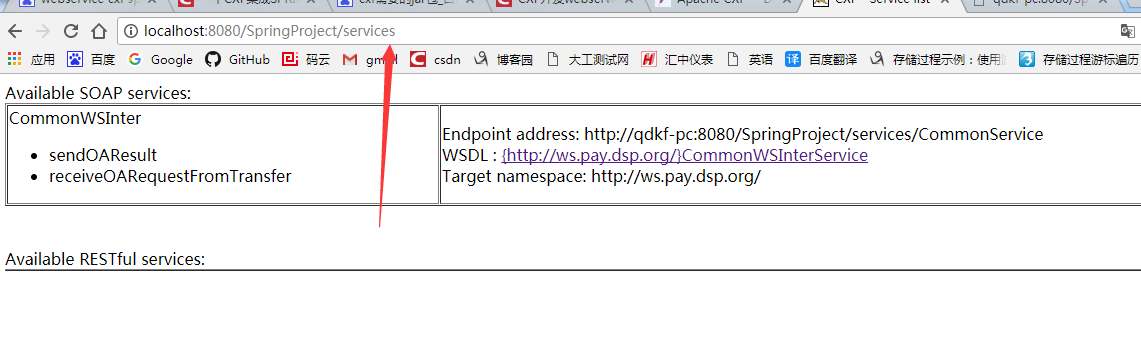
|  |
| --- |
| <context:annotation-config />  <context:component-scan base-package=*"com"* />  <context:component-scan base-package=*"com.hlj.springAnnotationTestSuccess"* />    <import resource=*"classpath:/module/applicationContext-cxf.xml"* />      </beans> |

## 2、先建立一个cxf的配置文件信息（我这里是从中科软复制过来的）

|  |
| --- |
| <?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"* xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:tx=*"http://www.springframework.org/schema/tx"* xmlns:cxf=*"http://cxf.apache.org/core"*  xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"* xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:policy=*"http://cxf.apache.org/policy"* xmlns:wsa=*"http://cxf.apache.org/ws/addressing"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://cxf.apache.org/bindings/soap http://cxf.apache.org/schemas/configuration/soap.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-3.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*  *http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd*  *http://cxf.apache.org/policy http://cxf.apache.org/schemas/policy.xsd*  *http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd*  *http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*  *"*>  <!-- 引cxf的一些核心配置 ，只要引入了cxf的jar包，这里就有效果了 -->  <import resource=*"classpath:META-INF/cxf/cxf.xml"* />  <import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"* />  <import resource=*"classpath:META-INF/cxf/cxf-extension-soap.xml"* /> |

# 3、web.xml中添加cfx的配置信息 ，service用法看下图

|  |
| --- |
| <!--2、 CXF配置 ， 所有/services请求都会先经过cxf框架-->  <servlet>  <description>Apache CXF EPRK</description>  <display-name>CXFServlet</display-name>  <servlet-name>CXFServlet</servlet-name>  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>  <load-on-startup>2</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>CXFServlet</servlet-name>  <url-pattern>/services/\*</url-pattern>  </servlet-mapping> |



# 4、建立service接口

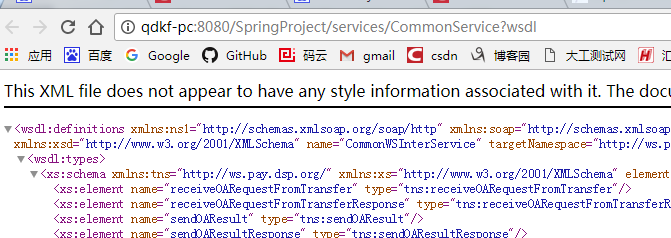
|  |
| --- |
| @WebService(targetNamespace="http://healerjean/")  **public** **interface** CommonWSInter {    **public** String receiveOARequestFromTransfer(@WebParam(name = "name") String name);    } |

# 5、添加实现接口

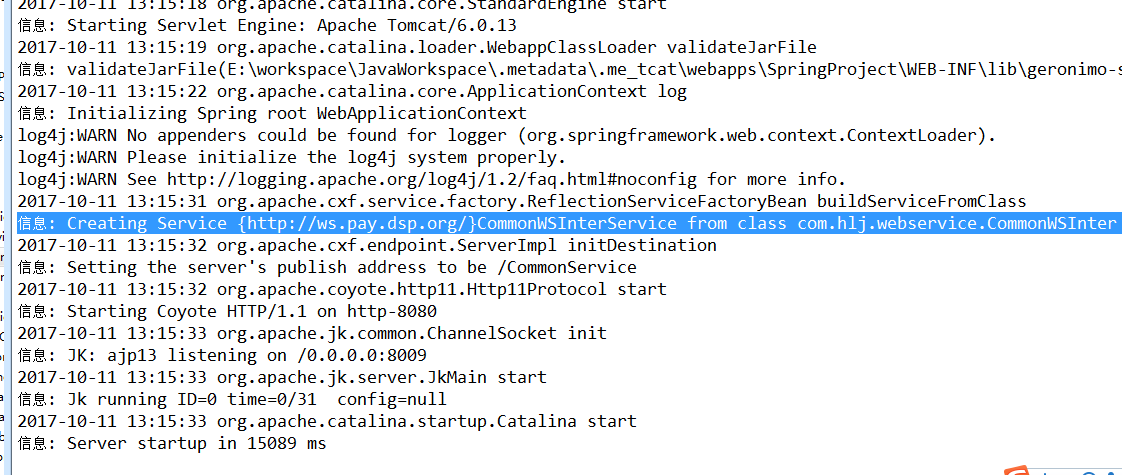
|  |
| --- |
| **public** **class** CommonWSImpl **implements** CommonWSInter{  @Override  **public** String receiveOARequestFromTransfer(String name) {  System.*out*.println("receiveOARequestFromTransfer 方法名字打印");    **return** "receiveOARequestFromTransfer";  } |

# 6、cxf配置spring文件中发布终端

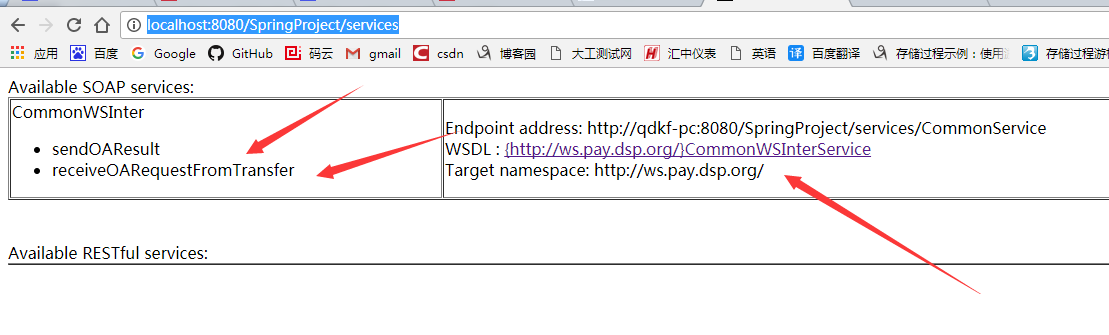
|  |
| --- |
| <!-- 引cxf的一些核心配置 -->  <import resource=*"classpath:META-INF/cxf/cxf.xml"* />  <import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"* />  <import resource=*"classpath:META-INF/cxf/cxf-extension-soap.xml"* />    <!-- 接口 -->  <jaxws:server id=*"commonWS"* serviceClass=*"com.hlj.webservice.CommonWSInter"*  address=*"/CommonService"*> 解释下图  <jaxws:serviceBean>  <ref bean=*"CommonWSImpl"* /> <!-- 引入接口实现类 -->  </jaxws:serviceBean>  </jaxws:server>  <!-- 接口实现类 -->  <bean id=*"CommonWSImpl"* class=*"com.hlj.webservice.CommonWSImpl"* /> |



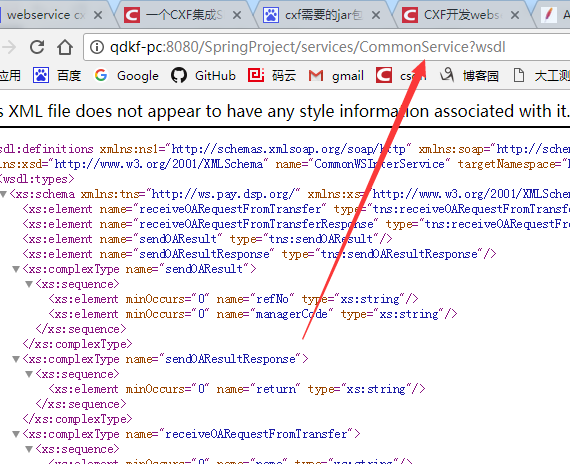
# 7、测试，run运行这个项目



## 1、地址栏中输入<http://localhost:8080/SpringProject/services>



## 2、点击这个链接，打开



# 二、建立客户端，自己调用自己（或者可以通过配置ip模拟调用其他人）

## 1、建立一个客户端的client-beans.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:sec=*"http://cxf.apache.org/configuration/security"*  xmlns:http=*"http://cxf.apache.org/transports/http/configuration"*  xmlns:p=*"http://cxf.apache.org/policy"*  xmlns:jaxws=*"http://java.sun.com/xml/ns/jaxws"*  xmlns:cxf=*"http://cxf.apache.org/core"*  xsi:schemaLocation=*"*  *http://cxf.apache.org/configuration/security*  *http://cxf.apache.org/schemas/configuration/security.xsd*  *http://cxf.apache.org/transports/http/configuration*  *http://cxf.apache.org/schemas/configuration/http-conf.xsd*  *http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd*  *http://cxf.apache.org/policy http://cxf.apache.org/schemas/policy.xsd*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.0.xsd"*>    <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-extension-http-binding.xml"*/>  <import resource=*"classpath:META-INF/cxf/cxf-extension-xml.xml"*/>  <import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"* />  <!--  <import resource="classpath:META-INF/cxf/cxf-extension-jaxws.xml" />  -->  <http:conduit name=*"\*.\*"*>  <http:client ConnectionTimeout=*"0"* ReceiveTimeout =*"0"* AutoRedirect=*"true"* Connection=*"Keep-Alive"*/>  </http:conduit>    </beans> |

## 2、实现客户端配置

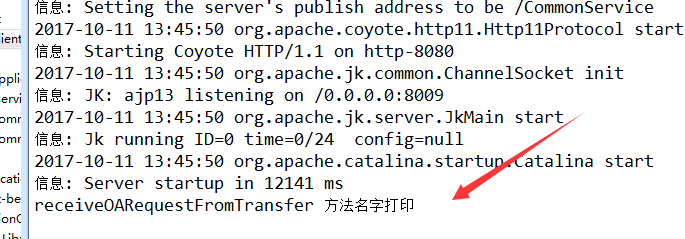
|  |
| --- |
| <bean id=*"commonClientFactory"* class=*"org.apache.cxf.jaxws.JaxWsProxyFactoryBean"*>  <property name=*"serviceClass"* value=*"com.hlj.webservice.CommonWSInter"* />  <property name=*"address"* value=*"http://localhost:8080/SpringProject/services/CommonService"* />  <!--<property name="address" value="http://10.3.181.48:8080/dsp\_oa/services/CommonService" /> -->  </bean>    <bean id=*"commonClient"* class=*"com.hlj.webservice.CommonWSInter"*  factory-bean=*"commonClientFactory"* factory-method=*"create"* /> |

## 3、开始测试

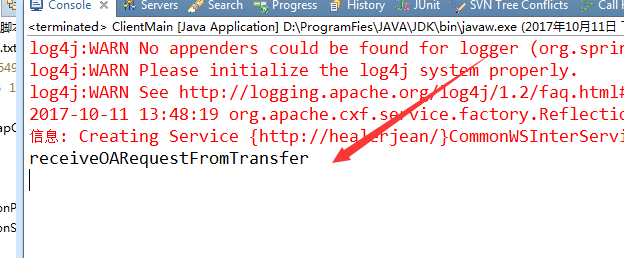
|  |
| --- |
| **public** **class** ClientMain {  **public** **static** **void** main(String[] args) {    ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("module/client-beans.xml");    CommonWSInter client = (CommonWSInter) context.getBean("commonClient");    String a = client.receiveOARequestFromTransfer("zhang");  System.*out*.println(a);  }  } |

## 4、控制台打印（需要启动tomcat服务）

### 1、tomcat控制台



### 2、main函数控制台



# 三、拦截器拦截报文（入报文和出报文，测试中科软报文日志入库）

## 1、建立入拦截器（对方调用函数时）

|  |
| --- |
| **public** **class** MyInInterceptor **extends** AbstractPhaseInterceptor<Message> {  **private** **static** **final** Logger *log* = Logger.*getLogger*(MyInInterceptor.**class**);    **public** MyInInterceptor() {  //这儿使用receive，接收的意思  **super**(Phase.*RECEIVE*);  }    **public** **void** handleMessage(Message message){    **try** {    InputStream is = message.getContent(InputStream.**class**);    String xml = IOUtils.*toString*(is);    **if**(is != **null**)  message.setContent(InputStream.**class**, is);  } **catch** (Exception e) {  *log*.error("Error when split original inputStream. CausedBy : "+"\n"+e);  } } |

## 2、建立出拦截器，方法调用完成。返回的结果进行拦截

|  |
| --- |
| **public** **class** MyOutInterceptor **extends** AbstractPhaseInterceptor<Message>{  **private** **static** **final** Logger *log* = Logger.*getLogger*(MyOutInterceptor.**class**);    **public** MyOutInterceptor() {  //这儿使用pre\_stream，意思为在流关闭之前  **super**(Phase.*PRE\_STREAM*);  }    **public** **void** handleMessage(Message message) {    **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);    //这里对xml做处理，处理完后同理，写回流中  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 {    }    } |

## 3、cxf的spring配置文件中添加上述自定义拦截器

|  |
| --- |
| <!-- CXF进入日志拦截器-->  <bean id=*"loggingInInterceptor"* class=*"org.apache.cxf.interceptor.LoggingInInterceptor"* />  <bean id=*"loggingOutInterceptor"* class=*"org.apache.cxf.interceptor.LoggingOutInterceptor"* />  <bean id=*"inMessageInteceptor"* class=*"com.hlj.MyInterceptor.MyInInterceptor"* />  <bean id=*"outMessageInteceptor"* class=*"com.hlj.MyInterceptor.MyOutInterceptor"* />    <cxf:bus>  <cxf:inInterceptors><ref bean=*"inMessageInteceptor"* /></cxf:inInterceptors>  <cxf:outInterceptors><ref bean=*"outMessageInteceptor"* /></cxf:outInterceptors>  </cxf:bus> |

## 4、开始利用map的进行测试

|  |
| --- |
| **public** **class** ClientMain {  **public** **static** **void** main(String[] args) {    ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("module/client-beans.xml");    CommonWSInter client = (CommonWSInter) context.getBean("commonClient");    Map<String,String> map = **new** HashMap<String,String> ();    map.put("name", "zhang");  map.put("xml", "xml");  map.put("age", "34");    String a = client.receiveOARequestFromTransfer("zhang");  client.sendMap(map);  System.*out*.println(a);  }  } |

## 5、控制台debug测试，观察上面的xml的值

### 1、如拦截器中取得的报文为

|  |
| --- |
| <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:sendMap xmlns:ns2="http://healerjean/"><arg0>  <entry>  <key>age</key>  <value>34</value>  </entry>  <entry>  <key>name</key>  <value>zhang</value>  </entry>  <entry>  <key>xml</key>  <value>xml</value>  </entry>  </arg0></ns2:sendMap></soap:Body></soap:Envelope> |

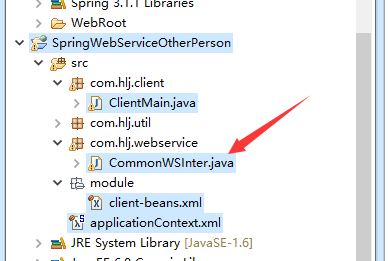
#### 、message为

|  |
| --- |
| {org.apache.cxf.message.Message.PROTOCOL\_HEADERS={Accept=[\*/\*], cache-control=[no-cache], connection=[keep-alive], Content-Length=[323], content-type=[text/xml; charset=UTF-8], host=[localhost:8080], pragma=[no-cache], SOAPAction=[""], user-agent=[Apache CXF 2.5.2]}, HTTP\_CONTEXT\_MATCH\_STRATEGY=stem, http.service.redirection=null, org.apache.cxf.request.url=http://localhost:8080/SpringProject/services/CommonService, org.apache.cxf.request.uri=/SpringProject/services/CommonService, HTTP.REQUEST=org.apache.catalina.connector.RequestFacade@79e59f7c, HTTP.CONFIG=org.apache.catalina.core.StandardWrapperFacade@5249c469, org.apache.cxf.transport.https.CertConstraints=null, Accept=\*/\*, org.apache.cxf.message.Message.PATH\_INFO=/SpringProject/services/CommonService, org.apache.cxf.message.Message.BASE\_PATH=/SpringProject/services/CommonService, org.apache.cxf.message.Message.IN\_INTERCEPTORS=[org.apache.cxf.transport.https.CertConstraintsInterceptor@68fdf3b3], org.apache.cxf.binding.soap.SoapVersion=org.apache.cxf.binding.soap.Soap11@788aa29e, org.apache.cxf.message.Message.ENCODING=UTF-8, org.apache.cxf.message.Message.QUERY\_STRING=null, HTTP.RESPONSE=org.apache.catalina.connector.ResponseFacade@55ba70e5, org.apache.cxf.security.SecurityContext=org.apache.cxf.transport.http.AbstractHTTPDestination$2@3de5da76, org.apache.cxf.request.method=POST, org.apache.cxf.async.post.response.dispatch=true, org.apache.cxf.configuration.security.AuthorizationPolicy=null, org.apache.cxf.message.MessageFIXED\_PARAMETER\_ORDER=false, org.apache.cxf.transport.Destination=org.apache.cxf.transport.servlet.ServletDestination@2cccf2e0, http.base.path=http://localhost:8080/SpringProject, Content-Type=text/xml; charset=UTF-8, HTTP.CONTEXT=org.apache.catalina.core.ApplicationContextFacade@435d671a} |

### 2、出拦截器中取得的报文为 与上面类似

# 6、全局拦截器和自身接口的拦截器（这个名字是我命名的，请观察17.3）

# 四、模拟远程其他用户调用自己，重新建立一个项目，只保留接口类



### 1、测试类不变

|  |
| --- |
| **public** **class** ClientMain {  **public** **static** **void** main(String[] args) {    ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("module/client-beans.xml");    CommonWSInter client = (CommonWSInter) context.getBean("commonClient");    Map<String,String> map = **new** HashMap<String,String> ();    map.put("name", "zhang");  map.put("xml", "xml");  map.put("age", "34");    String a = client.receiveOARequestFromTransfer("zhang");  client.sendMap(map);  System.*out*.println(a);  } |

### 2、调用成功，另外的项目服务端有反应

# 5、接3（拦截器）中科软拦截器

## 解释：根据soap报文里面的东西可以知道ip以及其他的信息，可以作为日志入库，以及可以对进入或者出去的异常进行采集，入库

## 1、入拦截器

|  |
| --- |
| **public** **class** InMessageInteceptor **extends** AbstractLoggingInterceptor {    **public** InMessageInteceptor() {  **super**(Phase.*RECEIVE*); // 必须在 READ 阶段前，否则消息 payload 获取不到  limit = -1; // 不进行报文长度限制  }    **public** **void** handleMessage(Message message) **throws** Fault {  **if** (message.getExchange().getOutMessage() == **null**) {  /\* 进入请求消息 \*/  WsContext.*getWsContext*().setContextId(IdUtil.*uuid32*()); // 事件起点，分配上下文标识  String payload = getMessagePlayload(message); // 消息负载  HttpServletRequest httpServletRequest = (HttpServletRequest) message.get("HTTP.REQUEST");  WsContext.*getWsContext*().setHttpServletRequest(httpServletRequest);    /\*  //InMessageEO zhge  InMessageEO inMessageEO = wsService.saveInMessage(  WsContext.getWsContext().getContextId(),  httpServletRequest  .getRemoteAddr(),  Channel.Webservice.getCode(),  InOutType.Request.getCode(),  payload,  null,null);  WsContext.getWsContext().setInRequestMessage(inMessageEO);\*/    //插入数据库zhon  System.*out*.println(  WsContext.*getWsContext*().getContextId()+  httpServletRequest  .getRemoteAddr()+  Channel.*Webservice*.getCode()+  InOutType.*Request*.getCode()+  payload);  }    }      **protected** String getMessagePlayload(Message message) {  //默认为1  String id = (String)message.getExchange().get(LoggingMessage.*ID\_KEY*);  **if** (id == **null**) {  id = LoggingMessage.*nextId*();  message.getExchange().put(LoggingMessage.*ID\_KEY*, id);  }  message.put(LoggingMessage.*ID\_KEY*, id);  **final** LoggingMessage buffer  = **new** LoggingMessage("Inbound Message\n----------------------------", id);  Integer responseCode = (Integer)message.get(Message.*RESPONSE\_CODE*);  **if** (responseCode != **null**) {  buffer.getResponseCode().append(responseCode);  }  //编码格式 encoding --- utf8  String encoding = (String)message.get(Message.*ENCODING*);  **if** (encoding != **null**) {  buffer.getEncoding().append(encoding);  }  //http 请求方法 post or get 这是get  String httpMethod = (String)message.get(Message.*HTTP\_REQUEST\_METHOD*);  **if** (httpMethod != **null**) {  buffer.getHttpMethod().append(httpMethod);  }  //message 内容类型 text/xml; charset=UTF-8  String ct = (String)message.get(Message.*CONTENT\_TYPE*);  **if** (ct != **null**) {  buffer.getContentType().append(ct);  }  Object headers = message.get(Message.*PROTOCOL\_HEADERS*);  **if** (headers != **null**) {  buffer.getHeader().append(headers);  }  //http://localhost:8080/SpringWebService/services/CommonService  String uri = (String)message.get(Message.*REQUEST\_URL*);  **if** (uri != **null**) {  buffer.getAddress().append(uri);  String query = (String)message.get(Message.*QUERY\_STRING*);  **if** (query != **null**) {  buffer.getAddress().append("?").append(query);  }  }    InputStream is = message.getContent(InputStream.**class**);  **if** (is != **null**) {  CachedOutputStream bos = **new** CachedOutputStream();  **try** {  IOUtils.*copy*(is, bos);  bos.flush();  is.close();  message.setContent(InputStream.**class**, bos.getInputStream());  writePayload(buffer.getPayload(), bos, encoding, ct);    **return** buffer.getPayload().toString();  } **catch** (Exception e) {  **throw** **new** Fault(e);  } **finally** {  **try** {  bos.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  **return** "";  }  @Override  **protected** Logger getLogger() {  **return** **null**;  }    /\*\*  \* 请求调用前（进入拦截器链上发生错误）异常响应处理  \*/  @Override  **public** **void** handleFault(Message message) {  **if** (message.getExchange().getOutMessage() == **null**) {  /\* 请求调用前（进入拦截器链上发生错误）异常响应处理 \*/  Exception e = message.getContent(Exception.**class**);  HttpServletRequest httpServletRequest = (HttpServletRequest) message  .getExchange().getInMessage().get("HTTP.REQUEST");  String stackTrace = ExceptionUtil.*getStackTraceAsString*(e);  InMessageEO inMessageEO = WsContext.*getWsContext*().getInRequestMessage();      /\* OutMessageEO outMessageEO = wsService.saveOutMessage(  WsContext.getWsContext().getContextId(),  httpServletRequest.getRemoteAddr(),  Channel.Webservice.getCode(),  InOutType.Response.getCode(),  stackTrace,  inMessageEO.getPkInMessage(),  RespStatus.Exception.getCode());  WsContext.getWsContext().setOutResponseMessage(outMessageEO);\*/    System.*out*.println(  WsContext.*getWsContext*().getContextId()+  httpServletRequest.getRemoteAddr()+  Channel.*Webservice*.getCode()+  InOutType.*Response*.getCode()+  stackTrace+  inMessageEO.getPkInMessage()+  RespStatus.*Exception*.getCode());  }  } |

## 2、出拦截器

|  |
| --- |
| **public** **class** OutMessageInteceptor **extends** AbstractLoggingInterceptor {    **public** OutMessageInteceptor(){  **super**(Phase.*PRE\_STREAM*);  limit = -1; // 不进行报文长度限制  }    @Override  **public** **void** handleMessage(Message message) **throws** Fault {  **if** (message.getExchange().getInMessage() != **null**) {  /\* 出去响应消息 \*/  **final** OutputStream os = message.getContent(OutputStream.**class**);  **if** (os == **null**) {  **return**;  }    **final** CacheAndWriteOutputStream newOut = **new** CacheAndWriteOutputStream(os);  message.setContent(OutputStream.**class**, newOut);  newOut.registerCallback(**new** LoggingCallback(message, os));  }    }    @Override  **public** **void** handleFault(Message message) {  **if** (message.getExchange().getInMessage() != **null**) {  /\* 出去响应消息，处理调用或出去拦截器链上发生的异常 \*/  InMessageEO inMessageEO = WsContext.*getWsContext*().getInRequestMessage();  Exception e = message.getContent(Exception.**class**);  String stackTrace = ExceptionUtil.*getStackTraceAsString*(e);  HttpServletRequest httpServletRequest = (HttpServletRequest) message  .getExchange().getInMessage().get("HTTP.REQUEST");      /\* OutMessageEO outMessageEO = wsService.saveOutMessage(  WsContext.getWsContext().getContextId(),  httpServletRequest.getRemoteAddr(), //ip地址  Channel.Webservice.getCode(), //表明为webservice服务  InOutType.Response.getCode(), //响应还是请求  stackTrace, //异常信息  inMessageEO.getPkInMessage(), //入烂机器的主键  RespStatus.Exception.getCode());// //表明是异常    WsContext.getWsContext().setOutResponseMessage(outMessageEO);  \*/  System.*out*.println(  WsContext.*getWsContext*().getContextId()+  httpServletRequest.getRemoteAddr()+  Channel.*Webservice*.getCode()+  InOutType.*Response*.getCode()+  stackTrace+  inMessageEO.getPkInMessage()+  RespStatus.*Exception*.getCode());  }  }    **class** LoggingCallback **implements** CachedOutputStreamCallback {  **private** **final** Message message;  **private** **final** OutputStream origStream;    **public** LoggingCallback(**final** Message msg, **final** OutputStream os) {  **this**.message = msg;  **this**.origStream = os;  }  **public** **void** onFlush(CachedOutputStream cos) {  }    **public** **void** onClose(CachedOutputStream cos) {  String id = (String)message.getExchange().get(LoggingMessage.*ID\_KEY*);  **if** (id == **null**) {  id = LoggingMessage.*nextId*();  }  **final** LoggingMessage buffer  = **new** LoggingMessage("Outbound Message\n---------------------------",  id);    Integer responseCode = (Integer)message.get(Message.*RESPONSE\_CODE*);  **if** (responseCode != **null**) {  buffer.getResponseCode().append(responseCode);  }    String encoding = (String)message.get(Message.*ENCODING*);  **if** (encoding != **null**) {  buffer.getEncoding().append(encoding);  }  String httpMethod = (String)message.get(Message.*HTTP\_REQUEST\_METHOD*);  **if** (httpMethod != **null**) {  buffer.getHttpMethod().append(httpMethod);  }  String address = (String)message.get(Message.*ENDPOINT\_ADDRESS*);  **if** (address != **null**) {  buffer.getAddress().append(address);  }  String ct = (String)message.get(Message.*CONTENT\_TYPE*);  **if** (ct != **null**) {  buffer.getContentType().append(ct);  }  Object headers = message.get(Message.*PROTOCOL\_HEADERS*);  **if** (headers != **null**) {  buffer.getHeader().append(headers);  }  **try** {  writePayload(buffer.getPayload(), cos, encoding, ct);  } **catch** (Exception ex) {  //ignore  ex.printStackTrace();  }    String payload = buffer.getPayload().toString();  HttpServletRequest httpServletRequest = (HttpServletRequest) message  .getExchange().getInMessage().get("HTTP.REQUEST");  //插入数据库    /\*  OutMessageEO outMessageEO = wsService.saveOutMessage(  WsContext.getWsContext().getContextId(),  httpServletRequest.getRemoteAddr(),  Channel.Webservice.getCode(),  InOutType.Response.getCode(),  payload,  inMessageEO.getPkInMessage(),  RespStatus.Normal.getCode());  WsContext.getWsContext().setOutResponseMessage(outMessageEO);  \*/  System.*out*.println(  WsContext.*getWsContext*().getContextId()+  httpServletRequest  .getRemoteAddr()+  Channel.*Webservice*.getCode()+  InOutType.*Request*.getCode()+  payload);    **try** {  cos.lockOutputStream();  cos.resetOut(**null**, **false**);  } **catch** (Exception ex) {  ex.printStackTrace();  }  message.setContent(OutputStream.**class**, origStream);  }  }  @Override  **protected** Logger getLogger() {  **return** **null**;  }  } |

## 3、其他相关类

### 1、WsContext统筹全局的类，用来包装其他对象

|  |
| --- |
| **public** **class** WsContext {    /\* 进入的请求消息 \*/  **public** **static** **final** String *IN\_REQ\_MESSAGE* = "IN.REQ.MESSAGE";    /\* 出去的响应消息 \*/  **public** **static** **final** String *OUT\_RESP\_MESSAGE* = "OUT.RESP.MESSAGE";    /\* 应用ContextPath \*/  **public** **static** **final** String *REQUEST\_BASEPATH* = "REQUEST.BASEPATH";    /\* 上下文标识 \*/  **public** **static** **final** String *CONTEXT\_ID* = "CONTEXT.ID";    **private** **static** ThreadLocal<WsContext> *threadLocal* = **new** ThreadLocal<WsContext>() {  **protected** WsContext initialValue() {  **return** **new** WsContext(**new** HashMap<String, Object>());  }  };    **private** **static** ServletContext *servletContext*;  **private** HttpServletRequest httpServletRequest;  **private** Map<String, Object> context;    **private** WsContext(Map<String, Object> context) {  **this**.context = context;  }  **public** **static** WsContext getWsContext() {  **return** (WsContext) *threadLocal*.get();  }    **public** String getContextId() {  **return** (String) get(*CONTEXT\_ID*);  }    **public** **void** setContextId(String contextId) {  put(*CONTEXT\_ID*, contextId);  }  **public** String getBasePath() {  HttpServletRequest request = getHttpServletRequest();  **if** (request == **null**) {  // 请求对象不存在则从context中查找  String basePath = (String) get(*REQUEST\_BASEPATH*);  **if** (basePath != **null** && basePath.trim().length() > 0) {  **return** basePath;  }  **return** **null**;  }  **return** request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + request.getContextPath() + "/";  }    **public** String getRemoteHost() {  **if** (getHttpServletRequest() == **null**) {  **return** **null**;  }  **return** getHttpServletRequest().getRemoteHost();  }                    **public** **void** setInRequestMessage(InMessageEO inMessageEO) {  put(*IN\_REQ\_MESSAGE*, inMessageEO);  }    **public** InMessageEO getInRequestMessage() {  **return** (InMessageEO) get(*IN\_REQ\_MESSAGE*);  }    **public** **void** setOutResponseMessage(OutMessageEO outMessageEO) {  put(*OUT\_RESP\_MESSAGE*, outMessageEO);  }    **public** OutMessageEO getOutResponseMessage() {  **return** (OutMessageEO) get(*OUT\_RESP\_MESSAGE*);  }    **public** Object get(String key) {  **return** context.get(key);  }  **public** **void** put(String key, Object value) {  context.put(key, value);  }    **public** **void** remove(String key) {  context.remove(key);  }    **public** ServletContext getServletContext() {  **return** *servletContext*;  }  **public** **void** setServletContext(ServletContext servletContext) {  WsContext.*servletContext* = servletContext;  }    **public** **synchronized** **static** **void** setAttributeForServletContext(String key, Object value) {  WsContext.*getWsContext*().getServletContext().setAttribute(key, value);  }  **public** **synchronized** **static** Object getAttributeFromServletContext(String key) {  **return** WsContext.*getWsContext*().getServletContext().getAttribute(key);  }    **public** HttpSession getHttpSession() {  **if** (httpServletRequest == **null**) {  **return** **null**;  }  **return** httpServletRequest.getSession();  }  **public** **void** setAttributeForSessionContext(String key, Object value) {  **if** (WsContext.*getWsContext*().getHttpSession() != **null**) {  WsContext.*getWsContext*().getHttpSession().setAttribute(key, value);  }  }    **public** Object getAttributeFromSessionContext(String key) {  **if** (WsContext.*getWsContext*().getHttpSession() == **null**) {  **return** **null**;  }  **return** WsContext.*getWsContext*().getHttpSession().getAttribute(key);  }  **public** HttpServletRequest getHttpServletRequest() {  **return** httpServletRequest;  }  **public** **void** setHttpServletRequest(HttpServletRequest httpServletRequest) {  **this**.httpServletRequest = httpServletRequest;  }    } |

### 2、Channel枚举、本服务是什么服务，当有jms的时候，日志进行区分

|  |
| --- |
| **public** **enum** Channel{  *Webservice*("WebService", "WebService"),    *JMS*("JMS", "JMS");  **private** String code;  **private** String label;  **private** Channel(String code, String label) {  **this**.code = code;  **this**.label = label;  } |

### 3、InOutType枚举、请求还是响应

|  |
| --- |
| **public** **enum** InOutType {  *Request*("Request", "请求"),    *Response*("Response", "响应");  **private** String code;  **private** String label;  **private** InOutType(String code, String label) {  **this**.code = code;  **this**.label = label;  } |

### 4、异常梳理的工具类

|  |
| --- |
| **public** **class** ExceptionUtil {  /\*\*  \* 将CheckedException转换为UncheckedException  \* **@param** e  \* **@return**  \*/  **public** **static** RuntimeException toUncheckedException(Exception e) {  **if** (e **instanceof** RuntimeException) {  **return** (RuntimeException) e;  } **else** {  **return** **new** RuntimeException(e);  }  }  /\*\*  \* 将StackTrace转化为String  \* **@param** e  \* **@return**  \*/  **public** **static** String getStackTraceAsString(Exception e) {  StringWriter stringWriter = **new** StringWriter();  e.printStackTrace(**new** PrintWriter(stringWriter));  **return** stringWriter.toString();  }    /\*\*  \* 获取异常消息  \* **@param** e  \* **@param** length，小于等于 0 时，不进行长度限制  \* **@return**  \*/  **public** **static** String getExceptionMessage(Exception e, Integer length) {  String msg = **null**;  **if** (e **instanceof** NullPointerException) {  msg = "java.lang.NullPointerException";  } **else** {  msg = e.getMessage();  }  **if** (length == **null** || length <= 0) {  **return** msg;  }  **if** (msg.length() > length) {  msg = msg.substring(0, length);  }  **return** msg;  }  } |

### 5、UUid工具类

|  |
| --- |
| **public** **class** IdUtil {  **private** IdUtil() {    }    /\*\*  \* uuid方式生成唯一标识，含“-”分隔符号，长度36位  \* **@return**  \*/  **public** **static** String uuid36() {  **return** UUID.*randomUUID*().toString();  }    /\*\*  \* uuid方式生成唯一标识，不含“-”分隔符号，长度32位  \* **@return**  \*/  **public** **static** String uuid32() {  **return** UUID.*randomUUID*().toString().replaceAll("-", "");  }    } |

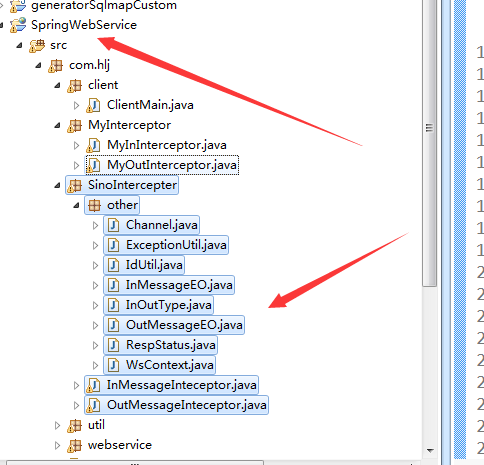
### 6、InMessageEO 入日志表

|  |
| --- |
| **public** **class** InMessageEO {  **private** String pkInMessage;  **private** String pkMessageInfo;  **private** String contextId;  **private** String messageFrom;  **private** String channel;  **private** String inOutType;  **private** String pkOutMessage;  **private** String memo;  **private** Date ts;  **private** String respStatus; |

### 7、OutMessageEO出日志表

|  |
| --- |
| **public** **class** OutMessageEO {  **private** String pkOutMessage;  **private** String pkMessageInfo;  **private** String contextId;  **private** String messageTo;  **private** String channel;  **private** String inOutType;  **private** String pkInMessage;  **private** String respStatus;  **private** String memo;  **private** Date ts; |

## 4、代码位置



# 6、java类进行调用