



Rcloud

产品用户开发手册

(RCloud 云 Exchange-Integration 服务)

北京中软国际信息技术有限公司

2010 年

目 录

第一章 概述.....	4
1.1 面向读者.....	4
1.2 开发准备.....	4
第二章 快速入门.....	4
2.1 基于消息交换的企业应用集成.....	5
2.1.1 应用场景.....	5
2.1.2 消息流设计.....	5
2.1.3 系统开发及部署.....	9
2.2 基于 IN-OUT 交换的企业应用集成.....	19
2.2.1 应用场景.....	19
2.2.2 消息流设计.....	19
2.2.3 系统开发与部署.....	19
第三章 开发指导.....	21
3.1 消息流设计.....	21
3.1.1 输入输出节点.....	21
3.1.2 普通节点.....	22
3.1.3 其它节点.....	22
3.1.4 消息流节点的使用.....	23
3.1.5 多输入消息流.....	24
3.1.6 消息流分支.....	25
3.1.7 迁移线条件设计.....	26
3.1.8 输入节点设置 XPATH 和消息格式.....	27
3.1.9 消息格式设计.....	28
3.1.10 消息映射.....	29
3.2 连接器开发.....	29
3.2.1 连接器类型.....	29

3.2.2 DEAgent 连接器.....	30
3.2.3 FTP 连接器.....	45
3.2.4 文件连接器.....	48
3.2.5 HTTP 连接器开发.....	49
3.2.6 SOAP/WS 连接器开发.....	50
3.2.7 IBM-MQ 连接器.....	56
3.2.8 TongLINK/Q 连接器.....	56
3.2.9 Win32Agent 连接器.....	57
3.2.10 数据库触发器.....	78
3.3 别名路由.....	80
3.4 RCloud 云 EI 服务服务器间传输方式.....	80
3.4.1 传输方式中多 MQ 通道/TongLINKQ 连接的使用.....	81
3.5 定时任务.....	84
3.5.1 接口.....	84
3.5.2 方法说明.....	84
3.5.3 代码示例.....	84
3.5.4 启动消息流.....	85
3.6 消息流 JAVA 扩展节点.....	86
3.6.1 接口.....	86
3.6.2 方法说明.....	87
3.6.3 代码示例.....	87
3.7 消息流 SQL 扩展节点.....	88
3.8 消息流中 WEB 服务调用.....	88
3.8.1 Web Service 配置.....	88
3.8.2 服务参数映射.....	89
3.8.3 响应处理.....	90
3.8.4 发送/接收带附件的消息.....	90
3.9 消息流中长周期 WEB 服务调用节点.....	91

3.9.1 回调服务	91
3.9.2 超时处理	92
3.9.3 代码示例	92
3.10 消息流错误处理节点	93
3.10.1 代码示例	94
3.11 消息多目的地分发	95
3.11.1 多目的地地址指定规则	96
3.11.2 多目的地地址集合结构说明	99
3.11.3 代码示例	100
3.12 客户端提供地址集合解析 API 使用说明	102
3.12.1 API 用途	102
3.12.2 代码示例	102
3.13 数据包导入到 RCloud 云 EI 服务服务器	103
3.14 消息跟踪服务接口	103
3.14.1 消息跟踪服务接口 API 列表	104
3.14.2 消息跟踪服务接口 API 使用说明	104
3.14.3 消息跟踪服务接口 API 详细说明	104
3.14.4 代码示例	111
3.15 服务运行状态和链路状态查询	114
3.15.1 接口 API	114
3.15.2 使用说明	114
第四章 开发建议	116
4.1 日志系统的使用	116
4.1.1 使用说明	116
4.1.2 方法说明	116
4.1.3 代码示例	117
4.2 数据源的使用	118
4.2.1 使用说明	118

4.2.2 方法说明	118
4.2.3 代码示例	118

第一章 概述

1.1 面向读者

本手册详细描述了 RCloud Exchange-Integration(以下简称 RCloud 云 EI 服务)的二次应用开发过程及核心 API 的使用方法,主要面向基于该产品的二次开发人员、实施部署人员。

1.2 开发准备

在开始进行二次应用开发之前,需要按照《RCloud4.5 Exchange-Integration 产品用户使用手册》手册部署 RCloud 云 EI 服务,使其满足下列先决条件:

1、完成 RCloud 云 EI 服务服务器的基本安装,包括 RCloud 云 EI 服务服务器,RCloud 云 EI 服务管理控制台,R1Proxy 应用,并且配置 RCloud 云 EI 服务服务器使其可以在 RCloud 云 EI 服务的管理控制台中被管理。

2、在 RCloud 云 EI 服务管理控制台中创建总线(单总线或复合总线),同时创建总线上的 RCloud 云 EI 服务服务器。

第二章 快速入门

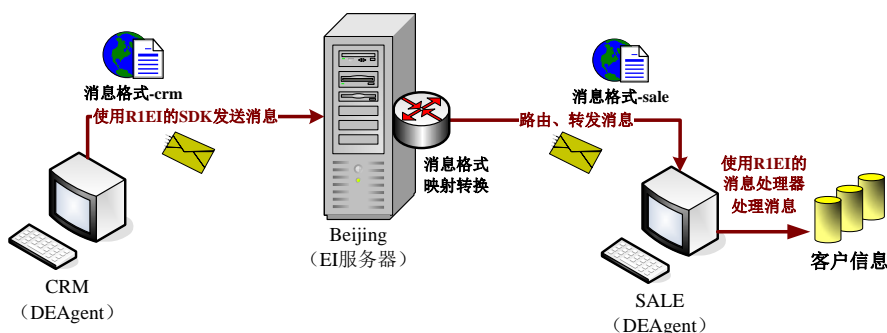
本节以典型的业务应用间数据信息交换过程为例,演示基于 RCloud 云 EI

服务的二次开发步骤与功能实现方式:

2.1 基于消息交换的企业应用集成

2.1.1 应用场景

现有客户关系管理系统（CRM）与销售任务管理系统（SALE）两个 WEB 应用，都通过 DEAgent 连接器与 RCloud 云 EI 服务服务器连接。其中 CRM 应用发送一条以新增客户信息为内容的消息，通过路由、计算、转换之后，SALE 应用能够接收该消息并在自身系统中同步新增该客户信息。

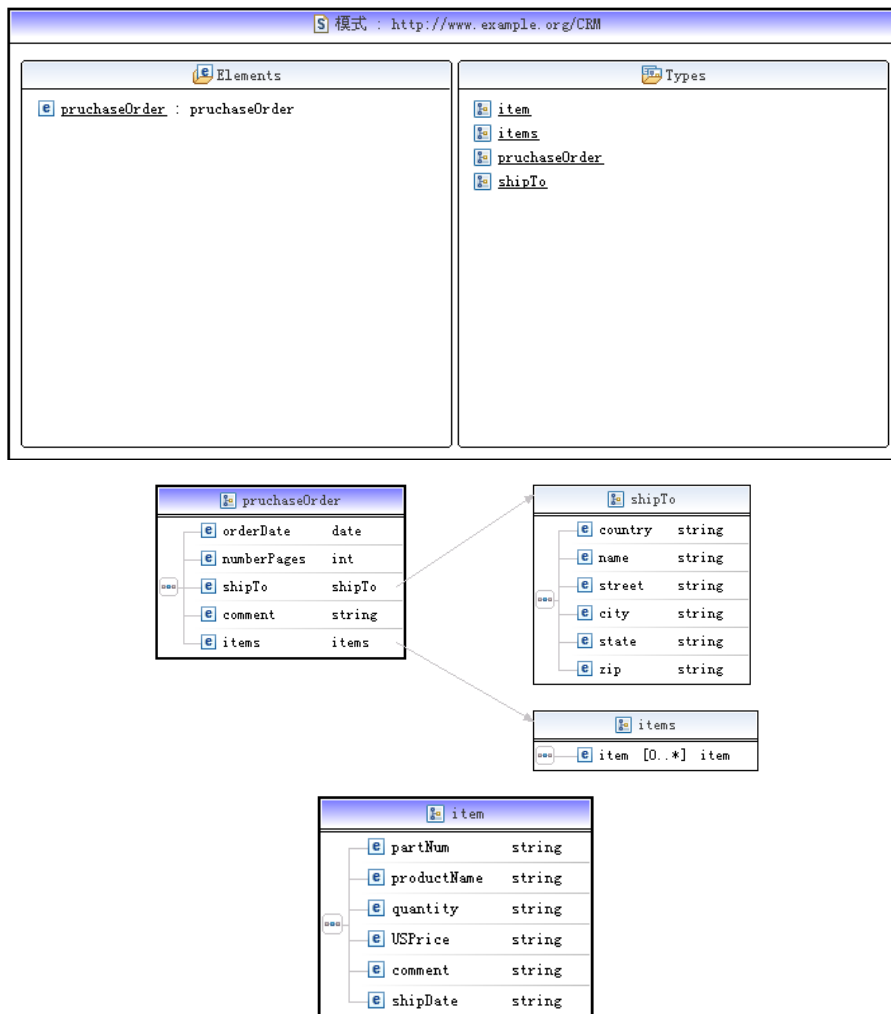


CRM 与 SALE 通过 RCloud 云 EI 服务进行消息交换示意图

2.1.2 消息流设计

2.1.2.1 创建发送消息的消息格式

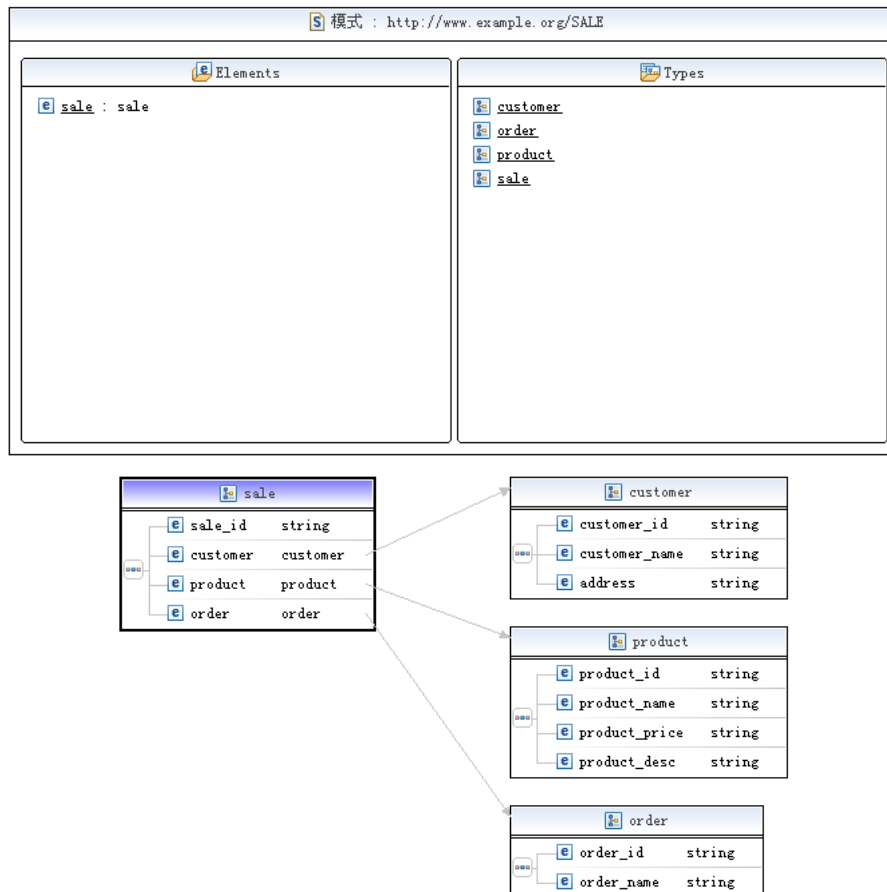
在 RCloud 云 EI 服务的设计工具中创建 CRM 应用发送的消息格式定义，如下图所示：



创建 CRM 消息

2.1.2.2 创建接收消息的消息格式

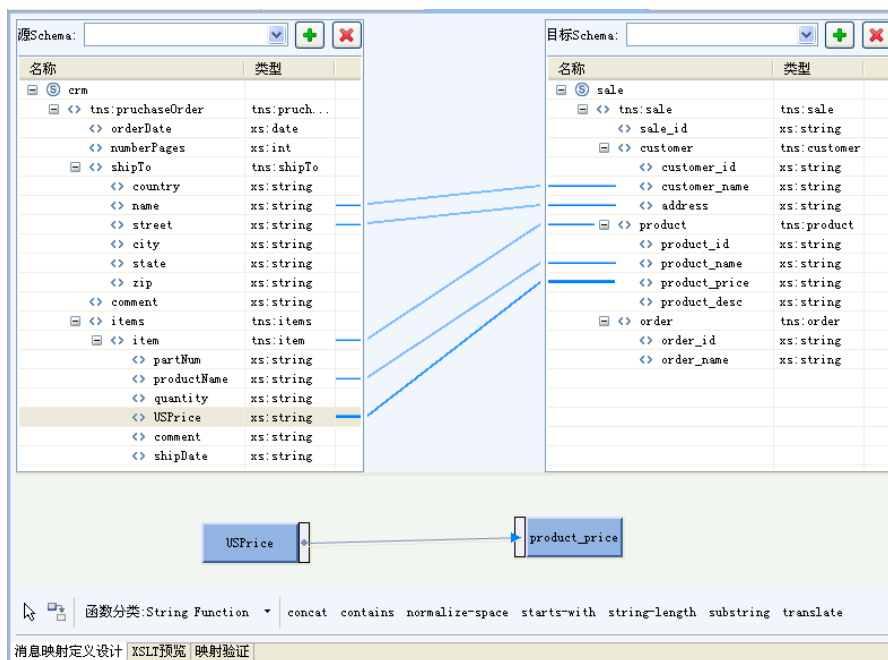
在 RCloud 云 EI 服务的设计工具中创建 SALE 应用接收的消息格式定义, 如下图所示:



创建 SALE 消息

2.1.2.3 创建消息映射

在 RCloud 云 EI 服务设计工具中创建 CRM 和 SALE 应用消息的消息映射，如下图所示：



创建消息映射

其映射关系的含义如下：

- 1、源的 `shipTo` 元素的子元素 `name` 的值赋给目标的 `customer` 的 `customer_name` 属性。
- 2、源的 `shipTo` 元素的子元素 `street` 的值赋给目标的 `customer` 的子元素 `address`。
- 3、源的 `items` 下面可能有多个 `item` 元素，有多少 `item` 元素，就产生多少 `product` 元素
- 4、源的每一个 `item` 元素的子元素的 `productName` 的值赋给目标相应的 `product` 的属性：`product_name`。
- 5、源的每一个 `item` 元素的子元素的 `USPrice` 的值赋给目标相应的 `product` 的属性：`product_price`。

2.1.2.4 创建消息流定义

在 RCloud 云 EI 服务设计工具中创建消息流定义，一个 DEAgent 输入节点，一个 DEAgent 输出节点，一个消息映射节点，如下图所示：



消息映射节点属性设置，选用刚才设计的消息映射定义，如下图所示：

消息映射节点配置

2.1.3 系统开发及部署

2.1.3.1 发送消息

使用 DEAgent 连接器客户端 API 发送消息，包含发送消息的演示页面和执行发送逻辑的 Servlet，页面如下图所示：

发送消息的代码片段如下：

```
String xmlMessage = request.getParameter("message");
R1DEClient rc = new R1DEClient();
MessageFlowHeader header = new MessageFlowHeader();
rc.setMessageHeader(header);
rc.setXmlMessage(xmlMessage);
rc.dispatchMessage();
```

2.1.3.2 接收消息

使用 DEAgent 连接器客户端 API 接收消息，只需要实现应用消息处理器即可，代码片段如下：

```
public class DemoReqMessageProcessor extends R1DEProcessor {
    private static final String module =
        WSInvokerTest.class.getName();
    public int processMessage(String messageStr) {
        //输出测试消息
        Debug.debug("\nIt's a test request result:"
            + "\n", module);
        Debug.debug("START*****START", module);
        Debug.debug(messageStr,module);
        Debug.debug("END*****END", module);
        return R1DEProcessor.PROCESS_RESULT_SUCCESS;
    }
}
```

2.1.3.3 开发发送消息和接收消息的应用

演示用的 CRM 应用和 SALE 应用都是 J2EE Web 应用，发送消息和接收消息的应用都是在一般的 web 应用基础上做一些改动，需要在开发过程中或开发完毕后，在一般的 web 应用中做一下几步：

1、将客户端部署压缩包(R1DEAgent4.X.X.zip)中 lib 和 3rdlib 下的所有 jar 文件复制到客户端应用的 /web-inf/lib 下。(4.x.x 表示该包的版本号)，将 R1DEAgent4.X.X.zip 中 classes 下的 rode.properties 文件复制到连接器所在应用的 /web-inf/classes 目录下。

2、将应用客户端部署压缩包中 wsdd 下的 server-config.wsdd 复制到客户端应用的/web-inf 下。

3、打开客户端应用的 web.xml，将下面的几部分内容添加到合适的位置。

```
<listener>
    <listener-class>org.apache.axis.transport.http.
        AxisHTTPSessionListener</listener-class>
</listener>
<servlet>
    <servlet-name>AxisServlet_DE</servlet-name>
    <display-name>DE Apache-Axis Servlet</display-name>
    <servlet-class>org.apache.axis.transport.http.
        AxisServlet</servlet-class>
    <init-param>
        <param-name>axis.servicesPath</param-name>
        <param-value>/deservices/</param-value>
    </init-param>
</servlet>
<servlet>        <servlet-name>SOAPMonitorService_DE</servlet-name>
    <display-name>DE SOAPMonitorService</display-name>
    <servlet-class>org.apache.axis.monitor.
        SOAPMonitorService</servlet-class>
    <init-param>
        <param-name>SOAPMonitorPort</param-name>
        <param-value>5001</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
</servlet>
<servlet>
    <servlet-name>DETServlet</servlet-name>
    <servlet-class>com.icss.ro.de.connector.httpimpl.
        DETServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>ClientInitServlet</servlet-name>
    <servlet-class>com.icss.ro.de.agent.admin.
```

```
ClientInitServlet</servlet-class>
    <load-on-startup>10</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>DETServlet</servlet-name>
    <url-pattern>/DETServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name> AxisServlet_DE</servlet-name>
    <url-pattern>/servlet/AxisServlet_DE</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name> AxisServlet_DE </servlet-name>
    <url-pattern>/deservices/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SOAPMonitorService_DE</servlet-name>
    <url-pattern>/deservices/SOAPMonitor</url-pattern>
</servlet-mapping>
```

```
<session-config>
    <session-timeout>5</session-timeout>
</session-config>
```

2.1.3.4 部署发送消息应用和接收消息应用

演示用的 CRM 应用和 SALE 应用都是 J2EE Web 应用,由于演示应用比较简单,可以部署到 Tomcat 上,无需创建和配置数据源等。

CRM 应用的上下文根设置为/crm, SALE 应用的上下文根设置为/sale。

然后,将发布包中客户端部署压缩包中 ResourceOneHome 文件夹下的内容全部拷贝到应用服务器所在盘(Windows 系统)或 root 根目录(UNIX 系统)下的 ResourceOneHome 目录下。将客户端部署压缩包中 ResourceOneHome/DataExchange 下的 rone.lic 文件,拷到 ResourceOneHome/license 下。

2.1.3.5 创建连接器

首先在 RCloud 云 EI 服务管理控制台中创建总线（名称为 **TestBus**），在总线中创建一个 EI 服务器节点（名称为 **Beijing**）。如下图所示：



总线及服务器节点配置

然后为 **CRM** 和 **SALE** 创建对应的集成对象 **CRM** 和 **SALE**，在集成对象中分别创建连接器 **crm** 和 **sale**。如下图：

集成对象类型: WEB应用	
集成对象名: CRM	集成对象简称: CRM
备注:	
<input type="button" value="确定"/> <input type="button" value="重置"/> <input type="button" value="取消"/>	

CRM 应用的集成对象

集成对象类型: WEB应用	
集成对象名: SALE	集成对象简称: SALE
备注:	
<input type="button" value="确定"/> <input type="button" value="重置"/> <input type="button" value="取消"/>	

SALE 应用的集成对象

基本信息	传输方式
连接器名称:	CRM连接器 OK
连接器ID:	crm OK
URL:	http://192.9.107.36:8080/crm OK
备注:	
下一步	

基本信息	传输方式
1. 连接器到DE-I服务器 HTTP/HTTPS	
起点:	crm
终点:	demoServer
URL:	http://192.9.107.37:9080/rode OK
2. DE-I服务器到连接器 HTTP/HTTPS	
起点:	demoServer
终点:	crm
URL:	http://192.9.107.36:8080/crm OK
上一步	

CRM 应用的连接器配置

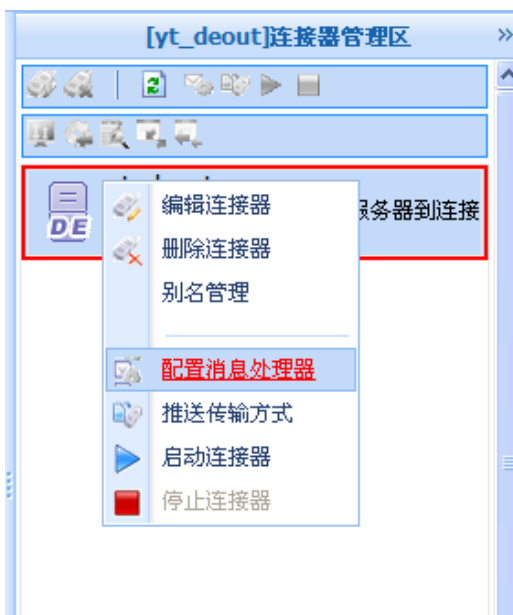
基本信息	传输方式
连接器名称:	SALE连接器
连接器ID:	sale
URL:	http://192.9.107.37:8080/sale
备注:	
下一步	

基本信息	传输方式
1. 连接器到DE-I服务器 HTTP/HTTPS	
起点:	sale
终点:	demoServer
URL:	http://192.9.107.37:9080/rode OK
2. DE-I服务器到连接器 HTTP/HTTPS	
起点:	demoServer
终点:	sale
URL:	http://192.9.107.39:8080/sale OK
上一步	

SALE 应用的连接器配置

SALE 应用的连接器还需要增加一个消息处理器，也就是前面开发的消息处理

器，如下图所示：

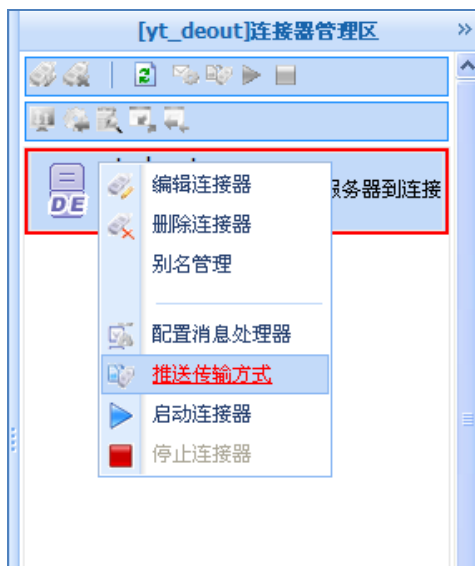


添加消息处理器

* ID	msgProcessor	OK	* 类型	普通消息处理器
* 名称	msgProcessor	OK		
* 类名	t.processor.DemoReqMessageProcessor	OK		
备注	<div style="border: 1px solid gray; height: 40px; width: 100%;"></div>			
	OK			
确定 清除 取消				

配置消息处理器

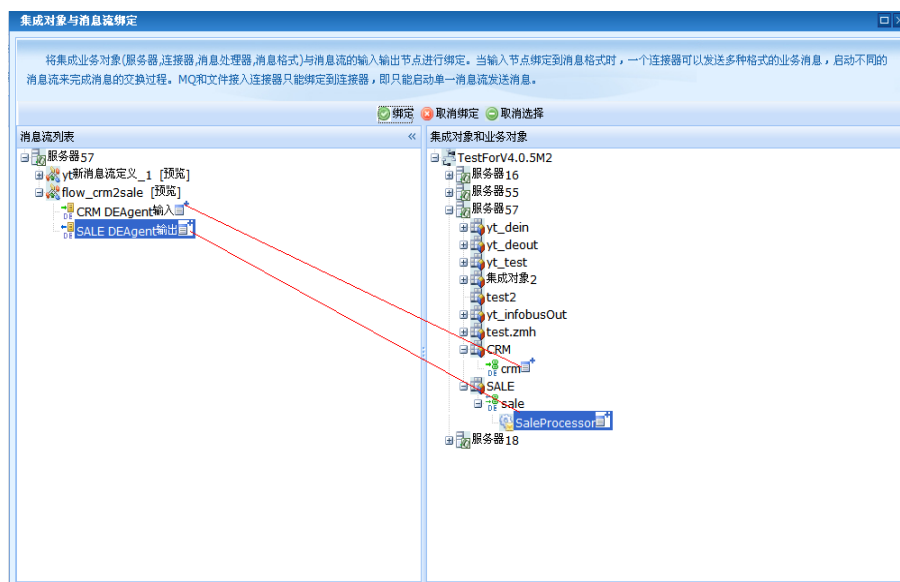
连接器配置编辑完成后，执行推送操作，将连接器配置分别推送到 CRM 和应用端。如下图所示：



推送传输方式

2.1.3.6 绑定连接器和消息流

连接器配置完成后，还需要和消息流进行绑定，才能使用消息流的路由和消息映射功能。将 CRM 应用的连接器和消息流的输入节点绑定，SALE 应用的消息处理器和消息流的输出节点绑定。绑定后，绑定操作区如下图所示：



绑定集成业务对象与消息流

配置完成后，重新启动 CRM 和 SALE 应用，并在 R1 管理控制台中启动连接器。

2.1.3.7 发送测试消息

在该示例中，我们使用如下的 XML 消息数据：

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseOrder orderDate="order_1" numberPages="8">
  <shipTo country="china">
    <name>ship_1</name>
    <street>xueyuan street beijing</street>
    <city>beijing</city>
    <state>1</state>
    <zip>100086</zip>
  </shipTo>
  <comment>Text</comment>
  <items>
    <item partNum="001">
      <productName>aigo MP3</productName>
      <quantity>80</quantity>
      <USPrice>2.8</USPrice>
      <comment/>
      <shipDate>2006-05-02</shipDate>
    </item>
    <item partNum="002">
      <productName>aigo DV</productName>
      <quantity>23</quantity>
      <USPrice>2.5</USPrice>
      <comment/>
      <shipDate>2006-05-11</shipDate>
    </item>
    <item partNum="003">
      <productName>Lenovo notebook</productName>
      <quantity>55</quantity>
      <USPrice>1.2</USPrice>
      <comment/>
      <shipDate>2006-03-30</shipDate>
    </item>
  </items>
</purchaseOrder>
```

访问 CRM 应用，发送测试消息的页面，将消息填入“消息内容”输入框中，点击发送按钮，发送消息的界面如下图所示：

消息内容:

```

<shipTo country="china">
  <name>ship_1</name>
  <street>xueyuan street
beijing</street>
  <city>beijing</city>
  <state>1</state>
  <zip>100086</zip>
</shipTo>
<comment>Text</comment>
</item>

```

发送

查看消息

发送消息界面

点击“查看消息”按钮，可以看到 CRM 应用发送的测试消息。如下图所示：

发送的请求消息列表：

消息序号	R1 DE-I服务器	发送时间	消息内容
0	s36	Thu Apr 17 16:41:12 CST 2008	ship_1 xueyuan street beijing beijing 1 100086 aigo MP3 80 2.8

访问 SALE 应用，查看测试消息的页面，可以看到接收到的消息。如下图所示：

接收到的请求消息列表：

消息序号	消息处理器	到达时间	消息内容
0	DemoReqMessageProcessor	Thu Apr 17 16:17:57 CST 2008	ship_1 xueyuan street beijing beijing 1 100086 aigo MP3 80 2.8

访问 RCloud 云 EI 服务的管理控制台，使用“消息跟踪”功能，也可以看到消息在 RCloud 云 EI 服务服务器上处理步骤，如下图所示：

返回

消息发送端	交换节点名称	消息接收端
crm ->		
	北京 ->	sale

消息交换路由路径信息列表

消息审计日志列表					
返回					
步骤	消息标识	消息类型	路由阶段	日志级别	时间
接收消息	1c519e2-1195b75...	请求	接入	正常	2008-04-17 16:17:42.859
消息流程启动	1c519e2-1195b75...	请求	流程	正常	2008-04-17 16:17:48.828
[CRM DEAgent输入]消息流活动执行	1c519e2-1195b75...	请求	流程	正常	2008-04-17 16:17:49.484
[消息映射]消息流活动执行	1c519e2-1195b75...	请求	流程	正常	2008-04-17 16:17:50.531
[SALE DEAgent输出]消息流活动执行	36b236b2-1195b7...	请求	流程	正常	2008-04-17 16:17:52.5
发送消息	36b236b2-1195b7...	请求	接出	正常	2008-04-17 16:17:53.078

消息审计日志列表

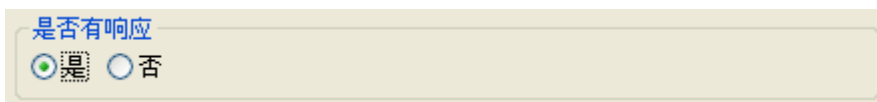
2.2 基于 In-out 交换的企业应用集成

2.2.1 应用场景

应用场景和 2.1.1 中大致相同，唯一的不同点为：SALE 应用接收到 CRM 应用所发的信息后，需要给 CRM 发送一条确认信息，告诉发送方已经成功接收信息。

2.2.2 消息流设计

消息流设计和 2.1.2 中基本相同，唯一的不同点为：消息流的输入节点中是否有响应框中需选中“是”。



2.2.3 系统开发与部署

涉及的应用有两个：CRM 应用和 SALE 应用，可以复用 2.1 中的资源

2.2.3.1 应用开发环境准备

和 [2.1.3.3](#) 中完全一致

2.2.3.2 发送消息

和 [2.1.3.1](#) 一致

2.2.3.3 接收消息

1. CRM 应用端用以接收响应消息的消息处理器开发

```
public class DemoReqMessageProcessor extends R1DEProcessor {
    private static final String module =
        WSInvokerTest.class.getName();
}
```

```

public int processMessage(String messageStr) {
    //输出测试消息
    Debug.debug("\nIt's a test request result:"
        + "\n",module);
    Debug.debug("START*****START",module);
    // TODO 用户对消息的处理代码
    Debug.debug(messageStr,module);
    Debug.debug("END*****END",module);
    return R1DEProcessor.PROCESS_RESULT_SUCCESS;
}
}

```

2. SALE 应用端用户接收请求消息并同时生成响应消息的消息处理器开发，需要重写方法 processAndResponse()

```

public class DemoReqMessageProcessor extends R1DEProcessor {

    private final String clientId =
        Router.getCurrentRouteNode().toString();

    public ResponseMessage
        processAndResponse(String messageStr) {

        // TODO 用户对请求消息的处理
        System.out.println(messageStr);
        // TODO 构造响应消息体
        ResponseMessage ps = new ResponseMessage();
        ps.setResponseStat(
            R1DEProcessor.PROCESS_RESULT_SUCCESS);
        ps.setResponseXml("<demoResult><title>
            It's response result. </title>" +
            "<abstract>Congratulations! It's OK.</abstract>"
            + "<timestamp>" + new Date().toLocaleString()
            + "</timestamp><responseFrom>"
            + clientId + "</responseFrom><content>"
            + messageStr + "</content></demoResult>");
        return ps;
    }
}

```

2.2.3.4 应用部署

和 [2.1.3.4](#) 一致

2.2.3.5 管控中创建连接器以及和消息流进行绑定

1. 连接器创建与消息处理器注册

与 [2.1.3.5](#) 一致，需要注意的是，与 CRM 应用相连的连接器上也需要注册用以处理响应消息的消息处理器。

2. 消息流绑定

与 [2.1.3.6](#) 一致，需要注意的是消息流中输入节点需要与 CRM 应用相连的连接器下的消息处理器绑定。

2.2.3.6 发送测试消息

和 [2.1.3.7](#) 一致

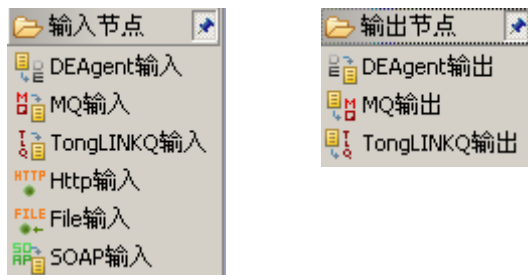
第三章 开发指导

3.1 消息流设计

在开发手册中，主要讲解 RCloud 云 EI 服务设计工具支持的几个重要的特性。

3.1.1 输入输出节点

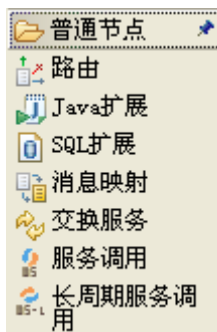
使用不同类型的连接器接入/接出 RCloud 云 EI 服务时，连接器对应的消息流的输入/输出节点也要使用和连接器类型一致的节点类型。如 DEAgent 连接器，对应的消息流输入节点类型就是 DEAgent。消息流支持的输入/输出节点类型如下所示：



消息流输入输出节点

3.1.2 普通节点

消息流中的普通节点是指介于输入节点和输出节点之间的业务功能节点，支持的节点如下图所示：

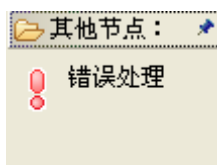


消息流普通节点

Java 扩展、SQL 扩展节点是提供给用户二次开发扩展的，在本章的 3.6 节和 3.7 节会详细介绍如何开发扩展。

服务调用、长周期服务调用节点都是调用用户提供的 Web 服务，可被调用的 Web 服务需要满足一定的规则，在本章的 3.8 节和 3.9 节会详细介绍如何开发扩展。

3.1.3 其它节点



消息流其它节点

错误处理节点是一种特殊的 Java 扩展节点，由用户扩展实现。用于处理消息

流节点中出现的异常情况、业务操作的事务补偿等目的。在本章的 3.10 节会详细介绍如何开发错误处理。

3.1.4 消息流节点的使用

消息流输入输出节点类型支持的中间节点表

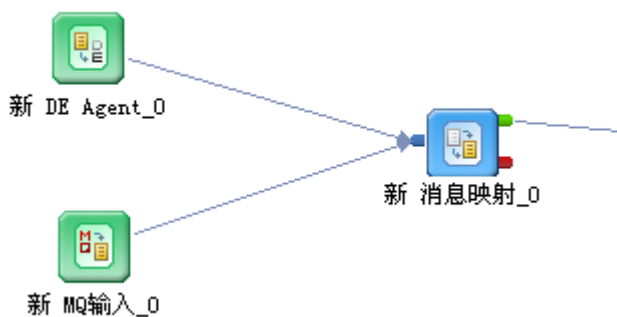
	支持响应	路由	Java 扩展	SQL 扩展	消息映射	服务调用	长周期服务调用	错误处理	交换服务	响应节点	DEAgent 输出	TongLINK /Q 输出	MQ 输出
DEAgent 输入	支持	√	√	√	√	√	√	√	√ 无响应	—	√	√	√
MQ 输入	否	√	√	√	√	√	√	√	√	—	√	√	√
TongLINK/Q 输入	否	√	√	√	√	√	√	√	√	—	√	√	√
HTTP 输入	否	√	√	√	√	√	√	√	√	—	√	√	√
File 输入	否	√	√	√	√	√	√	√	√	—	√	√	√
WEB 服务输入	支持	√	√	√	√	√	—	—	√ 无响应	√ 有响应	√ 无响应	√	√ 无响应
SOAP 输入	支持	√	√	√	√	√	√	√	√ 无响应	—	√	√	√

上表中的对应关系为消息流设计选择正确的节点选择提供了参考。

3.1.5 多输入消息流

在多个集成对象要协同完成一项工作时，可以使用多输入节点的消息流程，每个应用在消息流程中有对应的一个输入节点，多个输入节点可以是不同类型的，如【DEAgent】节点，【MQ 输入】节点等。

多个输入的同一笔业务使用关联键值来标识，如果多个应用发送到 RCloud 云 EI 服务服务器的消息的关联键值一样，就认为是同一笔业务的，多个消息会使用一个消息流程实例。多输入的流程如下图所示：



多输入流程

所有的输入节点都支持多输入，使用多输入的消息流时，客户应用发送的消息需要指定关联键，有些输入方式可以在提供的接入接口中指定；但所有的可以通过指定从消息内容中获取关联键的 XPATH 方式指定，如下图所示：

关联键值设置

多输入关联键值：

采用xpath格式，例如：/customer/name

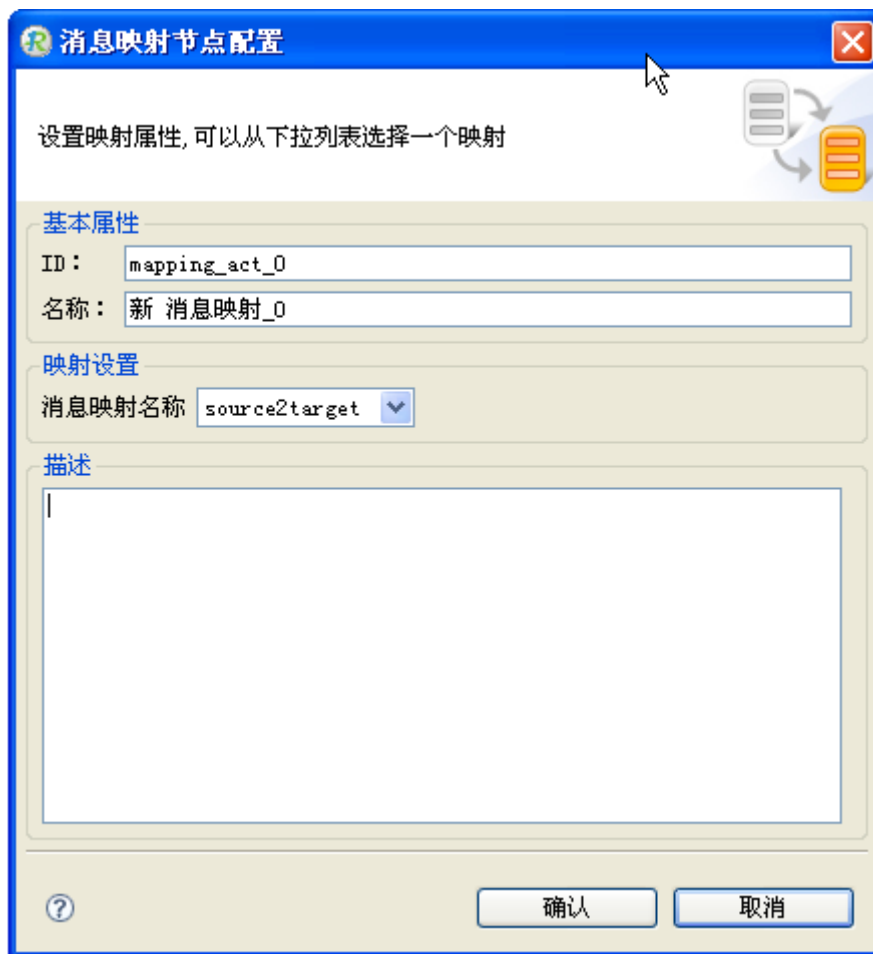
每个输入节点需要指定输入消息的消息格式（MessageSchema），如下图：

消息格式设置

☒ 消息格式

☐ 主题

而且输入节点指向消息映射节点，消息映射节点上设置使用哪个消息映射规则，如下图所示：

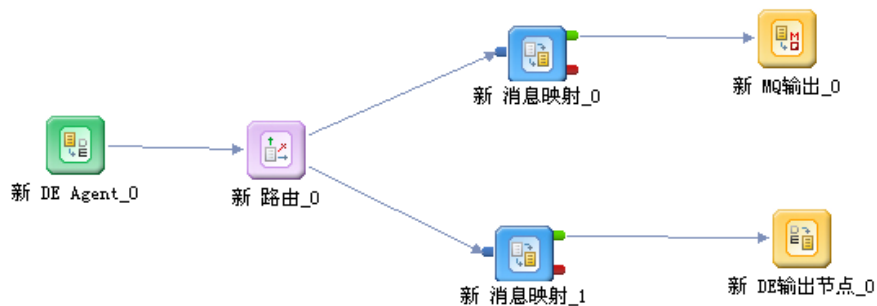


消息映射节点配置

消息格式和消息映射定义也是在 EI 设计工具中设计的。

3.1.6 消息流分支

当一个消息需要同时发送到多个目的地，或者从多个目的地中根据设置的迁移线条件动态确定发送到哪一个或者几个目的地时，可以使用带分支的流程，每个分支上都可以再添加分支，添加中间处理节点和输出节点，满足迁移线条件的目的地应用都会收到一份消息。可以在分支迁移线上设置条件，由 RCloud 云 EI 服务动态计算结果决定执行后面的哪条分支。



带分支的消息流

3.1.7 迁移线条件设计

迁移线条件表达式的条件类型有三种，分别是“xpath 简单组合语句”，“xpath 复杂组合语句”和“自定义 xpath 和 BeanShell 脚本”；前两种使用比较简单，参考使用手册即可。后一种可以支持复杂的 xpath 语句和 BeanShell 脚本，并且带有关键字着色和语法校验功能。利用 xpath 结果进行任何形式的计算。但需要遵循语法规则。例如：

迁移线属性

迁移线上可以设置符合BeanShell语法规则的表达式。

通用属性

ID: b7f2d0-118ca16f361-720ce6b9239ae896503f92110781ead0

名称: 迁移线

前驱: 新路由_0

后继: 新MQ输出_0

条件

条件类型: 条件

生成条件表达式: 自定义xpath和BeanShell表达式

验证

表达式:

```

xpath userName=/account/user_name;
for(int i=0;i<userName.size();i++){
    String uName=(String)userName.get(i);
    if(uName.indexOf("jim")>=0){
        return true;
    }
}
return false;
    
```

确认 取消

迁移线属性

语法规则如下:

1、如果使用 xpath, 则首先需要在脚本开始声明 xpath 变量, 以 xpath 开头, 变量名命名规则与 java 的相同。声明 xpath 变量语句每行均以分号“;”结尾, 而且每行只能声明一个变量, xpath 路径为全路径。

2、非 xpath 声明部分, 语法完全按照 bsh 的脚本语法, 返回值是字符串“true”或者布尔值 true, 或者非零的数字, 也认为条件成立。可以使用或者前面声明的 xpath 变量, 变量的数据类型均为 List 类型, List 的每个元素为 String 类型, 也就是根据 xpath 变量的 xpath 从 XML 消息中查询出的数据。

3.1.8 输入节点设置 XPATH 和消息格式

如果一个流程有多个输入节点, 那么就需要设置 Xpath, Xpath 查询到的数据

值作为多输入的关联键值。在多个应用客户端要协同完成一项工作（例如同一个消息流程）时，通过关联键值保证这些应用客户端进行同一项工作。例如，两个系统合作完成一项业务，这两个系统一定存在相同的属性值，通常是业务记录的主键 ID。如果流程是单输入的，那么就不需要设置 Xpath。

3.1.9 消息格式设计

消息支持的元素或者属性的数据类型如下：

Xml Schema 数据类型默认转换规则：

xml Schema 类型	转换后的 java.sql.Types 类型
integer	java.sql.Types.INTEGER
positiveInteger	java.sql.Types.INTEGER
negativeInteger	java.sql.Types.INTEGER
nonNegativeInteger	java.sql.Types.INTEGER
nonPositiveInteger	java.sql.Types.INTEGER
decimal	java.sql.Types.DECIMAL(16,3)
string	java.sql.Types.VARCHAR(255)
boolean	java.sql.Types.VARCHAR(255)
time	java.sql.Types.VARCHAR(255)
dateTime	java.sql.Types.VARCHAR(255)
duration	java.sql.Types.VARCHAR(255)
date	java.sql.Types.VARCHAR(255)
Name	java.sql.Types.VARCHAR(255)
QName	java.sql.Types.VARCHAR(255)
anyURI	java.sql.Types.VARCHAR(255)
ID	java.sql.Types.VARCHAR(255)
IDREF	java.sql.Types.VARCHAR(255)
NMTOKEN	java.sql.Types.VARCHAR(255)

对于上表中未提及的数据类型将不支持。

消息模型支持的数据类型为：

```
java.sql.Types.SMALLINT
java.sql.Types.INTEGER
java.sql.Types.BIGINT
java.sql.Types.FLOAT
java.sql.Types.REAL
java.sql.Types.DOUBLE
```

```
java.sql.Types.NUMERIC
java.sql.Types.DECIMAL
java.sql.Types.CHAR
java.sql.Types.VARCHAR
```

3.1.10 消息映射

消息流中的消息映射节点，可以对消息内容做合并，转换等处理。使用消息映射功能时，首先需要定义要转换的源消息的消息格式，和要得到的目标消息的消息格式，然后定义源格式和目标格式之间的转换关系，也就是消息映射；在定义消息流时，在消息流中增加消息映射节点，并选中使用刚才定义的消息映射。

3.2 连接器开发

3.2.1 连接器类型

RCloud 云 EI 服务提供的连接器类型包括：

1、DEAgent 连接器：既可以用作接入连接器，也可以用作接出连接器。提供 API 供客户应用调用，用户开发时只需要调用 API 即可将消息发送给 RCloud 云 EI 服务服务器，通过实现一个消息处理器就可以接收消息，开发简单，但需要将 API 的 jar 文件放入客户应用中，目前这种方式仅限基于 J2EE 平台开发的客户应用。

2、FTP 连接器：既可以用作接入连接器，也可以用作接出连接器。为大文件的传输提供支持，可以将客户应用产生的大文件从指定目录下传递到指定的 FTP 服务器上；也可以将 FTP 服务器上指定目录下的文件自动下载到本地主机指定目录下，但目前还不支持对文件目录的上传下载处理。这种连接器只需要在 RCloud 云 EI 服务的管理控制台中配置 FTP 连接器即可，不需要做代码开发。

3、文件连接器：只能用作接入。这种连接器支持将客户应用产生的数据文件根据用户设定的转换规则抽取为 XML 数据，通过 RCloud 云 EI 服务路由转换等处理后发送给目标应用；现在版本只支持 Microsoft Excel 97-2003 格式的(*.xls)文件。该种连接器也不需要做代码开发，但需要配置连接器并且设计、绑定消息

流。

4、HTTP 连接器：一种接入类型的连接器，支持客户应用通过向 RCloud 云 EI 服务服务器发送 HTTP/HTTPS 请求，将消息通过 RCloud 云 EI 服务服务器路由转换等发送给目标应用。

5、SOAP/WS 连接器：只能用作接入，但该连接器可以通过配置的 Web 服务接收反馈消息，该 Web 服务注册在 R1 统一资源库中，在设计消息流定义时的接入节点配置。

6、IBM-MQ 连接器：该种连接器可以从指定的 MQ 队列中自动读取消息，启动消息流，将消息路由到目标应用或者 MQ 队列。这种连接器也是只需要配置即可，无需做代码开发。

7、TongLINK/Q7 连接器：该类型的连接器的应用范围只针对 TongLINK/Q 7.0 的消息队列进行接入接出，从指定的 TongLINK/Q 队列中自动读取消息，启动消息流，将消息路由到目标应用或者 TongLINK/Q 队列。该连接器只需要配置即可，无需做代码开发。

8、win32Agent 连接器：该连接器是使用 delphi 开发的连接器。既可以用作接出连接器，也可以用作接入连接器。提供 API 供客户应用调用，用户开发时只需要调用 API 即可将消息发送给 RCloud 云 EI 服务服务器，可以接收消息，开发简单，但需要将 API 的 dll 文件放入客户应用中，目前这种方式仅支持以 ftp 的方式把数据传输到 EI 服务器，或者以 FTP 的方式获取消息数据。

3.2.2 DEAgent 连接器

DEAgent 连接器，由 RCloud 云 EI 服务提供 API，封装了发送/接收消息的逻辑。发送消息时，用户只需要调用 API；接收消息时，实现自己的消息处理器（通过实现 RCloud 云 EI 服务提供的接口类）即可；通过 DEAgent 发送消息时，消息可以带附件。

DEAgent 连接器发送消息时，目前支持两种方式：1）使用消息流绑定发送消息；2）直接指定目的地方式发送消息。

DEAgent 连接器通过消息流绑定方式发送消息时，需要先设计消息流，然后

将消息流和连接器绑定，每个接入连接器绑定一个接入节点，每个接出连接器的消息处理器绑定一个接出节点。

DEAgent 连接器使用直接指定目的地方式发送消息时，需要通过 SDK API 设置消息目的地地址以及消息处理器，然后发送消息。

DEAgent 连接器不仅支持单向的消息交换支持，也支持请求/反馈的消息交换（即消息接收端应用可以发送一个反馈消息给消息发送端）；使用消息流绑定方式时，可以在消息流的接入节点配置使用哪种模式，只有单输入节点且单输出节点的消息流才支持有反馈，使用请求/反馈的消息交换方式时，请求方的 DEAgent 也需要实现一个消息处理器来处理反馈消息；使用直接指定目的地时，需要通过 SDK API 设置响应消息处理器。

DEAgent 也支持多个连接器同时从同一个消息流接收消息；需要使用多输出的消息流。使用消息流绑定方式时，设计消息流定义时，对应每个接出连接器，定义一个接出节点，在管理控制台将每个接出节点与其对应的连接器的消息处理器绑定；使用直接指定目的地时，需要通过 SDK API 设置目的地地址集合（多个目的地地址）。

3.2.2.1 主要对象说明

3.2.2.1.1 MessageHeader

`com.icss.ro.de.agent.sdk.header.MessageHeader` 是负责包装消息头信息的抽象类，提供了一些公共属性的实现方法。

3.2.2.1.2 MessageFlowHeader

`MessageHeader` 的子类，增加了设置关联键值方法，用以标识一个消息的唯一性。

3.2.2.1.3 ReceivedMsgHeader

MessageHeader 的子类，用于使用消息处理器处理接收到的消息时，获取消息头信息，主要包含消息源信息。

3.2.2.1.4 R1DEClient

使用 com.icss.ro.de.agent.sdk.R1DEClient 客户进行应用开发时，调用此 API 发送消息。应用发送消息前，需设置相关的属性，如下：

- 1、消息头，包含消息传输所必需的信息，如果使用的单输入消息流程，而且发送的是请求消息，可以不设置；
- 2、消息内容，必填，消息内容支持 XML 格式的消息和纯文本格式的消息，对于纯文本格式的消息，不支持涉及消息格式相关的功能，比如 MAPPING 节点、WEB 服务调用节点；
- 3、消息附件，没有附件时不用设置。

3.2.2.1.5 FaultInfo

异常反馈消息对象，当消息在服务器端传输发生异常（例：网络不通等）后，如果发送方支持异常反馈，服务器端会构造一个异常反馈对象（FaultInfo），通知发送端应用。

发送端通过调用异常消息处理器的 processFaultInfo 方法来对异常反馈进行处理(参考 3.2.2.4.5 代码示例部分)

异常消息结构：

注意：元素<detail>下是一个 xml 格式的 String。

```
<soap-env:Fault
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  <faultactor></faultactor>
  <faultcode></faultcode>
  <faultstring></faultstring>
  <detail>
```

```
<DE_V4.0_DETAIL>
  <RelationKey>0101</RelationKey>
  <faultTime>Wed May 20 14:41:36 CST 2009</faultTime>
  <msgDestSet> Sichuan.gjj </msgDestSet>
  <faultMsgDestSet>Leshan.Sichuan.gjj</faultMsgDestSet>
  <faultMsg>error desc.....</faultMsg>
</DE_V4.0_DETAIL>
</detail>
</soap-env:Fault>
```

3.2.2.1.6 MessageAddressBean

地址信息封装类，通过提供的 `get` 方法，用户可以获取详细的地址信息，比如服务器 ID、连接器 ID、消息处理器 ID 等。

3.2.2.1.7 DestinationSet

消息目的地地址集合，包含一个或多个目的地路由信息。也可以是用模糊表达式指定的目的地。

详细说明请参考 [3.12 消息多目的地分发中](#) 相关说明

3.2.2.2 方法说明

3.2.2.2.1 MessageFlowHeader

- 1、获得消息关联键

```
public String getRelationKeyValue ()
```

- 2、设置消息关联键值

```
public void setRelationKeyValue(String relationKeyValue)
```

3.2.2.2.2 MessageHeader

- 1、设置消息 SchemaID

public void setMessageSchemald(String messageSchemald)

2、设置消息传输通道标识

public void setTransportChannel (String transportChannel)

3、设置消息处理器

public void setMsgProcessor(String processorId,String processorType)

4、指定目的地(nodeRouteName 和 nodeAlias 指定其中一个值即可， ClientID 和 Alias 也是指定其中一个值即可)，该方法和 setDestinations()互斥。

public void setDestination(String nodeRouteName,String nodeAlias ,
String clientID ,String clientAlias)

5、设置响应消息处理器(请求消息发送者 接收处理响应消息用)

public void setResponseMsgProcessor(String processorId)

6、设置消息处理类型

public void setMessageHandlerType(String messageHandlerType)

7、设置输入类型

public void setInputType(String inputType)

8、设置目的地集合，该方法和 setDestination ()互斥。

public void setDestinations(DestinationSet destset)

9、设置消息标题

public void setMessageTitle(String messageTitle)

3.2.2.2.3 ReceivedMsgHeader

1、获取消息格式 ID。

public String getMessageSchemald()

2、获取消息关联键值

public String getRelationKey()

3、获取消息发送端连接器 ID

public String getSourceClientId()

4、获取消息发送端 EI 服务器路由名称

```
public String getSourceServerRouteName()
```

3.2.2.2.4 R1DEClient

1. 设置消息头

```
public void setMessageHeader(MessageHeader messageHeader)
```

```
public void setMessageFlowHeader(MessageFlowHeader messageHeader)
```

2. 设置要发送的消息为响应消息类型,默认为无响应类型

```
public void setResponseMode ()
```

3. 设置 XML 格式文本作为消息体

```
public void setXmlMessage(String xmlMessage)
```

- 4、设置消息附件（文件）

```
public void addAttachmentsFile (File attachment)
```

- 5、设置消息附件（文件集合）， attachMap 的 key 为文件名， value 为文件对象

```
public void setAttachmentsFile (Map attachMap)
```

- 6、设置消息附件（流）

```
public void addAttachmentsStream(String name,InputStream in)
```

- 7、设置消息附件（流集合）， steamMap 的 key 为文件名， value 为附件流对象

```
public void setAttachmentsStream(Map steamMap)
```

- 8、异步发送消息

```
public void dispatchMessage()
```

- 9、导出消息到压缩文件

```
public String exportDataPackage (String path)
```

3.2.2.2.5 FaultInfo

1. 出现异常的消息的 relationKey

public String getRelationKey()

2. 异常发生时间

public String getFaultTime()

3. 异常发生的地址

public String getFaultMsgDestSet()

4. 出现异常的消息的目的地地址

public String getMsgDestSetStr()

5. 异常描述

public String getFaultMsg();

6. 获取出现异常消息的目的地地址,其中, exclude 地址是指 include 地址集合中需要排除的地址。

public MessageAddressBean[] getIncludeMsgDestSet();

public MessageAddressBean[] getExcludeMsgDestSet();

3.2.2.2.6 MessageAddressBean

1. 判断服务器地址是否是模糊指定

public boolean isServerExact()

2. 获取服务器地址类型, 分为路由名称和别名两种

public String getServerAddressType()

3. 获取服务器地址

public String getServerAddressValue ()

4. 判断连接器地址是否模糊指定

public boolean isClientExact()

5. 获取连接器地址类型, 分为连接器 ID 和别名两种

public String getClientAddressType ()

6. 获取连接器地址

public String getClientAddressValue ()

7. 判断消息处理器是否模糊指定

public boolean isProcessExact()

8. 获取消息处理器类型，分为消息处理器 ID 和 CLASS 两种

public String getProcessorType ()

9. 获取消息处理器

public String getProcessorValue ()

3.2.2.2.7 DestinationSet

1. 构造一个空的对象

public DestinationSet();

2. 使用地址集合 XML 字符串构造

public DestinationSet(String xml) **throws** XmlException

3. 在 include 集合中添加一个地址

```
public void includeAddress(String serverAddressType
    , String serverAddressValue
    , boolean serverAddressExact
    , String clientAddressType
    , String clientAddressValue
    , boolean clientAddressExact
    , String processorValueType
    , String processorValue
    , boolean processorValueExact)
```

4. 在 excluded 中添加一个地址（表示 include 中要排除的地址）

```
public void excludeAddress(String serverAddressType
    ,String serverAddressValue
    ,boolean serverAddressExact
    ,String clientAddressType
    ,String clientAddressValue
    ,boolean clientAddressExact)
```

3.2.2.3 EI 消息处理器

消息处理器用来在 DEAgent 连接器的客户应用端处理接收到的消息。当 RCloud 云 EI 服务服务器将消息发送给客户应用端的 DEAgent 时，DEAgent 会自动调用客户配置的消息处理器处理该消息。如果是需要发送反馈消息，消息处理器接口也提供了机制，只需要在实现类中覆盖相应的方法即可。消息处理器如何配置请参考使用手册。本小节主要介绍如何开发用户自己的消息处理器。

3.2.2.3.1 消息处理器接口

`com.icss.ro.de.agent.processor.R1DEProcessor` 是消息处理器抽象类，客户编写消息处理器时继承这个类，根据业务需求覆盖父类方法，完成处理消息的功能。方法如下：

1、处理接收到的 XML 消息，该方法适用于不需要返回结果消息给发送方的情况

```
public int processMessage(String messageStr)
```

参数 `messageStr` 为消息内容

返回值取值为下面的常量说明中的值

2、处理接收到的 XML 消息，并返回一个结果消息

```
public ResponseMessage processAndResponse(String messageStr)
```

参数 `messageStr` 为消息内容

返回值为 `ResponseMessage` 对象。

3、获取消息头信息

```
public ReceivedMsgHeader getMessageHeader()
```

4、取得消息附件列表

```
public Map getMessageAttachments()
```

5、根据附件名，取得附件

```
public File getMessageAttachment(String name)
```

6、删除所有接收消息的附件文件

```
public void deleteMessageAttachments()
```

7、删除指定名字的附件

```
public void deleteMessageAttachment(String name)
```

8、从消息中移除对指定名字的附件引用

```
public void removeMessageAttachment(String name)
```

9、处理接收到的错误消息

```
public int processFaultInfo(FaultInfo fault);
```

10、获取消息头中相关属性值

```
public String getHeaderProp(String key)
```

注意：消息处理器的加载是由 DEAgent 连接器运行时的一个独立线程加载的，独立于 ServletContext、ServletRequest、Session、filter、listener 等机制；所以当需要使用这些机制获取某些资源时（如由 Listener 初始化 context 的 Spring 去创建类实例等），就不能再从 ServletContext 等中读取信息，可以考虑将需要的信息直接从数据库/配置文件中读取，或者将信息缓存到某个静态类中，编写消息处理器逻辑时，从缓存中读取。

3.2.2.3.2 反馈消息

当消息处理器需要返回一个结果给发送方时，需要返回的结果需要包装为 ResponseMessage 对象。包含的方法如下：

1、设置 XML 格式字符串形式的消息体

```
public void setResultXml(String resultXml)
```

2、设置消息处理器处理的执行结果状态，参数为下面的常量说明中的值

```
public void setResultStat(int resultStat)
```

3、取得反馈消息的附件列表

```
public Map getAttachmentSet()
```

4、添加附件

```
public void addResultAttachment(File resultAttachment)
```


5、添加附件

public void addResultAttachments(List attachments)

attachments 为附件对象列表，对象为 File 类型

3.2.2.3.3 常量说明

1、适配器处理结果：成功

public final static int *PROCESS_RESULT_SUCCESS*

2、适配器处理结果：失败，且不可恢复

public final static int *PROCESS_RESULT_FAIL*

3、适配器处理结果：失败，但可以恢复

public final static int *PROCESS_RESULT_RECOVERABLE*

4、消息处理器类型——普通

public static final String *PROCESSOR_TYPE_GENERAL*

5、消息处理器类型——独立运行

public static final String *PROCESSOR_TYPE_INDEPENDENT*

3.2.2.3.4 异常反馈消息处理器

继承com.icss.ro.de.agent.processor.R1DEProcessor，重写**public int** processFaultInfo(FaultInfo fault) 方法。用户自定义对异常反馈的处理逻辑（参考3.2.2.4.5代码示例）。

异常消息处理器开发完成后，需要部署到客户端应用的/web-inf/classes 目录下；同时需要在管控中的连接器上注册该异常反馈消息处理器，并和连接器所在集成对象的业务类型关联。

注意：消息处理器必需和业务对象类型相关联，所以，发送的消息必需设置业务类型，同时，异常消息处理器注册的时候，需要和业务类型相绑定。

3.2.2.4 代码示例

3.2.2.4.1 发送消息的代码示例

1、DEAgent 通过绑定的消息流发送消息示例 1

```
MessageFlowHeader header = new MessageFlowHeader();
header.setRelationKeyValue ("messageRelationKey001");
//如果消息流中绑定 schemaId，则在消息头中需要设置 schemaId
header.setMessageSchemaId ("SCM_080604024627");
//如果用户需要指定专用的传输通道，则在消息头中需要设置传输通道标识
header.setTransportChannel ("tran_channel_1");
R1DEClient rc = new R1DEClient();
rc.setMessageHeader(messageHeader);
rc.setXmlMessage("<message>...</message>");
rc.dispatchMessage();
```

2、DEAgent 通过直接指定目的地方式发送消息示例 2

```
MessageFlowHeader header = new MessageFlowHeader();
header.setRelationKeyValue ("messageRelationKey001");
//设置目的地节点路由名称和连接器 ID
header.setDestination("Beijing.icss.com", null,
    " Beijing.agent.out.1", null);
//header.setDestination(nodeRouteName, nodeAlias, clientID, clientAlias) 目的地
路由名称(nodeRouteName)和别名(nodeAlias)二者任选其一，连接器 ID(clientID)
和连接器别名(clientAlias)二者任选其一即可完成对目的地的设定。对应的另一个
参数置为 null 即可。

//设置目的地客户端消息处理器 ID
header.setMsgProcessor ("processor_1",
    R1DEProcessor.PROCESSOR_TYPE_GENERAL);
//如果用户需要指定专用的传输通道，则在消息头中需要设置传输通道标识
header.setTransportChannel("tran_channel_1");
R1DEClient rc = new R1DEClient();
rc.setMessageHeader(messageHeader);
rc.setXmlMessage("<message>...</message>");
rc.dispatchMessage();
```

3、DEAgent通过指定目的地集合的方式发送消息

注意：使用目的地集合和使用指定单个目的地是互斥的，都指定了，则只按指定目的地集合的方式来处理；

代码示例参考[3.12.3代码示例](#)

3.2.2.4.2 接收消息的代码示例

```
public class ReceiveAndResponder extends R1DEProcessor {
    /* 处理接收到的消息*/
    public int processMessage(String messageStr) {
        Debug.debug("=====START");
        Debug.debug(messageStr);
        Debug.debug("=====END");
        return R1DEProcessor.PROCESS_RESULT_SUCCESS;
    }

    /* 处理接收到的消息，并发送反馈消息*/
    public ResponseMessage processAndResponse(
        String messageStr) {
        Debug.debug("~~~~~");
        Debug.debug(messageStr);
        Debug.debug("~~~~~");
        ResponseMessage ps = new ResponseMessage();
        ps.setResponseStat(
            R1DEProcessor.PROCESS_RESULT_SUCCESS);
        String resMsg="<demoResult>" +
            "<title>It's response result.</title>" +
            "<content>Congratulations! It's OK.</content>" +
            "<requestMessage>" +messageStr+
            "</requestMessage></demoResult>";
        ps.setResponseXml(resMsg);
        return ps;
    }
}
```

3.2.2.4.3 发送带附件消息的代码示例

```
String xmlMessage = "<a>....</a>";
R1DEClient rc = new R1DEClient();
```

```

MessageFlowHeader header = new MessageFlowHeader();
rc.setMessageHeader(header);
rc.setXmlMessage(xmlMessage);
// 设置附件 (文件格式)
rc.addAttachmentsFile(new File("c:\\demo.txt"));
// 设置附件 (流格式)
Rc.addAttachmentsStream("demo.txt",
new FileInputStream(new File("c:\\demo.txt")));
rc.dispatchMessage();

```

3.2.2.4.4 接收带附件消息的代码示例

```

public ResponseMessage processAndResponse(
    String messageStr) {
    ResponseMessage ps = new ResponseMessage();
    Map messageAttachments = this.getMessageAttachments();
    if(messageAttachments!=null){
        for(Iterator iter =
            messageAttachments.keySet().iterator();
            iter.hasNext();) {
            String element = (String)iter.next();
            //save write to I/O
            write("d:\\\" + element,
                messageAttachments.get(element));
        }
    }

    ps.setResponseStat(R1DEProcessor
        .PROCESS_RESULT_SUCCESS);
    //如果响应消息带附件，则可以添加附件，
    //不过消息发送方应该对应地建立消息处理器处理响应消息
    String resMsg=
        "<demo><re>It's responseresult.</re></demo>";
    ps.setResponseXml(resMsg);
    List attachments = new ArrayList();
    attachments.add(new File("d:\\result.doc"));
    ps.setResultAttachments(attachments);
    return ps;
}

```

3.2.2.4.5 异常反馈消息处理代码示例

```
public class DemoReqMessageProcessor extends R1DEProcessor {
    /**
     * 处理异常反馈消息
     */
    public int processFaultInfo(FaultInfo fault){
        // 发生异常的消息的唯一标识
        String rk = fault.getRelationKey();
        // 异常发生时间
        String time = fault.getFaultTime();
        // 异常发生地址
        String faultAddr = fault.getFaultMsgDestSet();
        // 发生异常的消息的目的地地址(字符串)
        String destAddr = fault.getMsgDestSet();
        // 异常描述信息
        String faultDesc = fault.getFaultMsg();

        // 消息目的地信息(MessageAddressBean)
        MessageAddressBean[] beans =
            fault.getIncludeMsgDestSet();
        MessageAddressBean[] beans =
            fault.getExcludeMsgDestSet();

        // TODO 对异常通知的后续处理，用户实现
        return R1DEProcessor.PROCESS_RESULT_SUCCESS;
    }
}
```

3.2.2.4.6 数据包导入到 EI 服务器代码示例

JSP 代码片段

```
<form name="form1" method="POST"
    action="http://localhost:9080/rode/ZipDataImportServlet"
    enctype="multipart/form-data">
    把zip数据文件包导入到EI服务器
<br>

<input type='FILE' name='upload_1' size='80'>
```

```
<input name="button" type="submit" value="导入EI服务器数据"
      onClick="setAction()"><br>

</form>
```

3.2.2.4.7 数据包导出

将需要发送的消息按 RCloud 云 EI 服务消息格式以 ZIP 形式导出到文件系统。

```
MessageFlowHeader header = new MessageFlowHeader();
header.setRelationKeyValue ("messageRelationKey001");
//设置目的地节点路由名称
header.setDestNodeId ("Beijing.icss.com");
header.setDestClientId ("Beijing.agent.out.1");
//设置目的地客户端消息处理器 ID
header.setMsgProcessor ("processor_1",
    R1DEProcessor.PROCESSOR_TYPE_GENERAL);
//如果用户需要指定专用的传输通道，则在消息头中需要设置传输通道标识
header.setTransportChannel ("tran_channel_1");
R1DEClient rc = new R1DEClient();
rc.setMessageHeader(messageHeader);
rc.setXmlMessage("<message>...</message>");
// 将消息打包导出，参数为文件导出的目录地址
rc.exportDataPackage("d:/exportdir")

.....
```

3.2.3 FTP 连接器

FTP 连接器可以支持大文件的双向传输，高效稳定；FTP 连接器也可以与其他连接器结对使用。FTP 连接器开发用户系统不需要调用我们提供的 API，只需要把文件放到指定目录下即可，下面章节给出了 Java 语言接入的片段。

3.2.3.1 基于 FTP 连接器开发用户系统

一、对于用户系统移动到的 FTP 目录里面的文件，FTP 连接器会自动处理。

FTP 连接器预留了文件类型接口*.tmp。当一个文件通过其他系统写入到 FTP 目录时，只要在传输的过程中文件后缀名为.tmp，连接器会认为是一个正在传输的文件不会做任何的处理。当文件传输完毕，用户系统可以将文件名改回正常格式，连接器即会进行处理。

Java 代码示例：

```
/*用户系统将文件接入到 FTP 目录示例 */
public static int sendMsg(String message) {
    // message 为要传入的带路径的文件名 如： d:/a/a.zip;
    String fileDir = "d:/test"; // fileDir 为 FTP 目录
    int tag = 0;
    try {
        // 对 fileDir 路径进行有效性的校验
        File uploadDir = new File(fileDir);
        if (!uploadDir.exists())
            uploadDir.mkdirs();
        File f = new File(message);
        String filename = f.getName();
        String tmpname;
        if (filename.indexOf(".") != -1) {
            // 没有后缀的文件名
            tmpname = filename.substring(0, filename.lastIndexOf("."));
        } else {
            tmpname = filename;
        }
        String remoteFileName = tmpname + ".tmp";
        File dest = new File(fileDir, remoteFileName);
        copyFile(message, dest.getPath());
        dest.renameTo(new File(fileDir, filename));
    } catch (Exception e) {
        tag = 0;
    }
    return tag;
}
public static void copyFile(String srcName, String tagName) {
    FileInputStream inputStream = null;
    BufferedInputStream inBuff = null;
```

```

        FileOutputStream outputStream = null;
        BufferedOutputStream outBuff = null;
        try {
            inputStream = new FileInputStream(srcName);
            inBuff = new BufferedInputStream(inputStream);
            outputStream = new FileOutputStream(tagName);
            outBuff = new BufferedOutputStream(outputStream);

            byte[] readBytes = new byte[2048];
            int len;
            try {
                while ((len = inBuff.read(readBytes)) != -1) {
                    outBuff.write(readBytes, 0, len);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {

            try {
                if (inputStream != null)
                    inputStream.close();
                if (inBuff != null)
                    inBuff.close();
                if (outputStream != null)
                    outputStream.close();
                if (outBuff != null)
                    outBuff.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

二、 FTP 连接器将消息文件在接出到 FTP 服务器时，会将一个正在传输的文件后缀改成*.tmp，处理完毕的时候会文件改成原来的后缀名。用户系统在 FTP 目录下检索文件的时候需要筛选文件后缀为.tmp 的文件不进行处理。

Java 代码示例:

```
/*FTP 接入对*.tmp 为后缀的文件进行过滤示例 */
public static int inputFile(){
File filePool = new File("d:/ftpPool");
if (!filePool.isDirectory())
return -1;
String[] fileNames = filePool.list();
if (fileNames == null)
return -1;
for (int i = 0; i < fileNames.length; i++) {
if (fileNames[i].endsWith(".tmp"))
continue;
// doMyWork(...);
}
return 1;
}
```

3.2.4 文件连接器

文件连接器是一种接入类型的连接器，它自动将客户的文件按照定制的“文件格式转换器”转换成 XML 格式，并发送到 RCloud 云 EI 服务服务器，当前只支持 Excel97-2003 格式的文件接入，可以从指定的目录中读取 Excel 文件，做文件数据格式转换生成 XML 消息，并自动发送到 RCloud 云 EI 服务服务器。

从 Excel 文件抽取转换成 XML 时，现在只支持文件格式转换，不支持对内容的处理（如使用函数变换值），如果需要对文件内容做处理，可以使用消息流的【消息映射】环节。

文件连接器也是需要在一个 Web 应用环境中运行。Web 应用可以是正在使用的 J2EE 平台的业务系统，也可以独立部署一个 J2EE Web 应用。

使用文件连接器首先要在 RCloud 云 EI 服务设计工具中设计“文件格式转换器”，将“文件格式转换器”发布到统一资源库后，可以在 RCloud 云 EI 服务的管理控制台配置一个该类型的连接器，并将配置信息推送到文件连接器客户应用端即可，不需要做代码开发。

如果使用多输入的消息流，那么需要在输入节点上配置 Excel 文件转换成的 XML 消息的消息格式，而且要指定获取关联键值的 XPATH。

3.2.5 HTTP 连接器开发

HTTP 连接器也是一种接入连接器，客户端应用通过向 RCloud 云 EI 服务发送标准的 HTTP 请求，并根据约定将一些参数放在请求中，就可以接入到 RCloud 云 EI 服务。该类型的连接器只支持向 RCloud 云 EI 服务发送消息，不支持从 RCloud 云 EI 服务接收消息；向 RCloud 云 EI 服务发送 HTTP 请求的代码需要用户自己开发，鉴于兼容不同的平台，RCloud 云 EI 服务不提供 API。

RCloud 云 EI 服务提供的接受 HTTP 请求的 URL 为：

`http://[RCloud 云 EI 服务服务器 IP]:[RCloud 云 EI 服务服务器 port]/[RCloud 云 EI 服务服务器应用上下文根]/HttpServlet`

其中[]中的内容要换为实际值，如果 RCloud 云 EI 服务服务器部署的容器使用 HTTPS 协议，URL 开头可以更换为 https，客户发送的 HTTP 请求也要做对 HTTPS 的支持。

HTTP 会话的 Response 的 STATUS 值，为 200 时表示消息发送成功，其它情况表示失败；如果状态值为 70，表示 RCloud 云 EI 服务接收消息时，接收到的参数不符合要求，这时需要检查下客户端发送的 HTTP 请求是否符合规则；如果状态值为 500，表示 RCloud 云 EI 服务服务器接收消息时，内部出现运行时异常，需要确认 HTTP 请求是否正确。

需要传递的参数如下：

参数 Key	必填	说明
messageBody_V4.0	是	要发送的消息内容，必须是 XML，必须传递
messageSourceClientId	是	注册的连接器 ID，必须传递
messageRelationKeyValue	否	关联键值，如果使用的是单输入的消息流，就不需要填；如果是多输入的消息流，需要填写，也可以使用另一种方式传递关联键值，参考【关联键】说明

messageSchemaId	否	业务消息类型(SchemaId), 消息流中所使用的消息格式
messageDestinationNodeId	否	目的地路由名称
messageDestinationClientId	否	目的地客户端 ID
messageDestinationNodeAlias	否	目的地服务器别名
messageDestinationClientAlias	否	目的地客户端别名
messageTransportChannel	否	传输链路的通道标识
messageHandlerId	否	消息处理器 ID, 当以指定目的地的方式发送消息, 并且接收端为 DEAgent 时需要指定
title	否	消息标题

3.2.6 SOAP/WS 连接器开发

SOAP/WS 连接器是一种接入类型的连接器, 支持接收异步反馈消息, 支持带附件的消息。客户应用通过访问 RCloud 云 EI 服务服务器提供的 Web 服务, 连接到 RCloud 云 EI 服务, 这个 Web 服务是注册在 RCloud 云 EI 服务服务器的服务列表中, 客户应用可以直接访问, 和访问消息流服务的方式一致。需要在 RCloud 云 EI 服务的管理控制台中注册一个 SOAP/WS 连接器。本小节主要介绍如何通过访问 SOAP/WS 连接器的 Web 服务。

3.2.6.1 Web 服务

RCloud 云 EI 服务注册在 R1 统一资源库中提供接入的支持的 Web 服务为:

- 1、服务名: WSConnectionService
- 2、版本: 1.0
- 3、命名空间: <http://webservice.connector.de.ro.icss.com/xsd>
- 4、服务方法: 返回是否接收成功 (true 表示接收成功, false 为接收失败)

/ 1-基本服务调用, 包括消息标题、关联键值、源客户端 ID、消息内容及业务消息类型 */*

```
public boolean receiveRequestMessage(String messageTitle,String keyValu,
String sourceClientId,String messageBody,String schemaId)
```

/ 2-指定传输通道的服务调用, 包括基本服务调用参数和传输链路标识 */*

```
public boolean receiveReqMsgWithChannel(String messageTitle,String
```

keyValue,

String sourceClientId,String messageBody,

String schemaId,String msgTransportChannel)

/* 3-指定目的地的服务调用，包括基本服务调用参数和目的地信息*/

public boolean receiveReqMsgWithDest(String messageTitle,String keyValue,

String sourceClientId,String messageBody,String schemaId,

String destNodeId,String destClientId,String messageAdaptId)

/* 4-指定目的地和传输通道的服务调用，包括基本服务调用参数、传输链路标识和目的地信息*/

public boolean receiveReqMsgWithDestChannel(String messageTitle,String

keyValue,

String sourceClientId,String messageBody,String schemaId,

String destNodeId,String destClientId,String msgTransportChannel,

String messageAdaptId)

/* 5-指定目的地别名的服务调用，包括基本服务调用参数、目的地别名信息*/

public boolean receiveReqMsgWithDestAlias(String messageTitle,String

keyValue,

String sourceClientId,String messageBody,String schemaId,

String destNodeAlias,String destClientAlias,String messageAdaptId)

/* 6-指定目的地别名和传输通道的服务调用，包括基本服务调用参数、传输链路标识和目的地别名信息*/

public boolean receiveReqMsgWithDestAliasChannel(String

messageTitle,String keyValue,

String sourceClientId,String messageBody,String schemaId,

String destNodeAlias,String destClientAlias, String msgTransportChannel,

String messageAdaptId)

参数说明如下：

参数名称	说明
messageTitle	消息标题
relationKey	关联键值，使用的消息流程为单输入方式时，单输入时可以传递空字符串；多

	输入时也可以使用 xpath 方式指定
sourceClientID	连接器 ID，必填
messageBody	XML 格式的消息内容，必填
schemald	设置业务消息类型 (schemald)
messageDestinationNodeid	目的地路由名称
messageDestinationClientId	目的地客户端 ID
messageDestinationNodeAlias	目的地服务器别名
messageDestinationClientAlias	目的地客户端别名
messageTransportChannel	传输链路的通道标识
messageHandlerId	消息处理器 ID，当以指定目的的方式发送消息，并且接收端为 DEAgent 时需要指定

3.2.6.2 有反馈的 Web 服务

如果客户应用需要接收异步反馈消息，还需要提供一个 Web 服务，并注册到 R1 统一资源库中，由 RCloud 云 EI 服务回调传递反馈消息；客户应用提供的 Web 服务的方法需要是两个参数，分别是：

String relationKey: 反馈消息的关联键值，根据它可找到是对应哪个请求消息的。

String messageBody: 反馈消息的内容。

该 Web 服务需要在消息流的接入节点配置，如下图所示：

是否有反馈

☒ 是 ☐ 否 设置当前服务输入节点是否需要输出节点反馈处理结果。

接受响应信息的Web服务配置

设置用于接受响应信息的web服务, 该服务必须由客户提供, 并已发布到统一资源库。

R1统一资源库:

Web服务名称: 例如: SaleDataService

Web服务版本: 例如: 1.0

Web服务命名空间: 例如: http://de.ro.icss.com/xsd

Web服务操作名: 例如: getSaleData

Web 服务接入节点配置

3.2.6.3 代码示例

1、通过 R1 统一资源库客户端获取注册的服务地址

```

/**
 * 通过 R1 统一资源库客户端获取注册的服务地址
 * @return
 */
public String gerServiceUrl(){
    //资源库地址
    String _grUrl = "http://192.9.107.16:9080/gr";
    //R1 统一资源库客户端
    GRClient client = new GRClient();
    client.setUrl(_grUrl);
    //资源名称
    String resName = "TestService";
    //资源版本号
    String resVersion = "1.0";
    return client.getEndpoint(resName,resVersion);
}

/**
 * 服务调用示例
 */
public void call(){
    try {
        String _url = gerServiceUrl();
        //命名空间，方法名
        QName _qName =
            new QName("http://your.namespace","method");
        //调用参数
        Object[] _opArgs = new Object[]{"arg0","arg1"};
        Class[] _returnTypes = new Class[]{String.class};
        RPCServiceClient serviceClient =
            new RPCServiceClient();
        Options options = serviceClient.getOptions();
        EndpointReference targetEPR =
            new EndpointReference(_url);
        options.setTo(targetEPR);
        //服务响应结果
        Object[] response = serviceClient
            .invokeBlocking(_qName, _opArgs,

```

```

        _returnTypes);
        Debug.debug("response:"+response);
    } catch (AxisFault e) {
        Debug.error(e,e.getMessage(),module);
    }
}

```

2、自定义方法解析服务地址

```

/* 服务调用示例 */
public Object[] invoke(String code, String version,
    String _ns, String _operate, Object[] para,
    Class[] returnType)throws WSInvokerException {
    if (code==null || "".equals(code.trim()) ||
        version==null || "".equals(version.trim()) ||
        _ns==null || "".equals(_ns.trim()) ||
        _operate == null || "".equals(_ns.trim())){
        throw new WSInvokerException(
            "invoke ws failed, Invoke parameter is null!");
    }
    // 获取注册到统一资源库的 WSDL
    byte[] bytes= accessRepository(code,version);
    if (bytes == null) {
        throw new WSInvokerException(
            "invoke ws failed, can not find wsdl file!");
    }
    try {
        Definition definition= getWSDefinition(bytes);
        String url=getUrl(definition);
        Debug.debug("ws invoker url:"+url,module);
        return callService(url, _ns, _operate,
            para, returnType);
    } catch (AxisFault e) {
        Debug.error(e,"invoke error.",module);
        throw new WSInvokerException("invoke ws failed!",e);
    } catch (Exception e) {
        Debug.error(e,"invoke error.",module);
        throw new WSInvokerException("invoke ws failed!",e);
    }
}

/* 从资源库获取 WSDL 内容 */
private byte[] accessRepository(String code,String version){

```

```

try{
    if (repositoryAddress == null) {
        throw new WsInvokerException(
            "repositoryAddress is null!");
    }

    // resource client
    Debug.debug("repository url:"+repositoryAddress
        ,module);
    ResourceClient r1client = new ResourceClient();
    r1client.setEndPoint(repositoryAddress);
    byte[] bytes = r1client.getResourceEntity(
        code, version);
    return bytes;
} catch (Exception e){
    Debug.error(e,"accessRepository error.",module);
}
return null;
}

/* 解析 WSDL 对象 */
private Definition getWSDefinition(byte[] bytes){
    WSDLFactory factory;
    try {
        factory = WSDLFactory.newInstance();
        WSDLReader WSDLReader = factory.newWSDLReader();
        InputSource ins = new InputSource(
            new ByteArrayInputStream(bytes));
        Definition definition = WSDLReader.readWSDL(
            null, ins);
        return definition;
    } catch (WSDLException e) {
        Debug.error(e,"getWSDefinition error.",module);
        return null;
    }
}

/* 从 WSDL 定义中得到服务访问地址 */
private String getUrl(Definition definition ){
    Map map = definition.getServices();
    String url = null;
    for (Iterator it = map.keySet().iterator();
        it.hasNext();) {
        QName name = (QName) it.next();
    }
}

```



```

        if(name.getNamespaceURI().equals(namesaceURL)){
            Service service = (Service) map.get(name);
            Map ports = service.getPorts();
            for (Iterator it1 = ports.keySet().iterator();
                it1.hasNext();) {
                String name1 = (String) it1.next();
                Port port = (Port) ports.get(name1);
                Iterator ita = port.
                    getExtensibilityElements().iterator();
                while (ita.hasNext()) {
                    SOAPAddressImpl ee = (SOAPAddressImpl)
                        ita.next();
                    url = (ee.getLocationURI());
                    break;
                }
            }
            break;
        }
    }
    return url;
}

```

3.2.7 IBM-MQ 连接器

IBM-MQ 连接器，即可以用作接入类型的连接器，也可以用作接出类型的连接器；用作接入类型的连接器时，可以从指定的 MQ 队列中读取消息，启动消息，用户只需要将消息放入指定的队列中即可；用作接出类型的连接器时，RCloud 云 EI 服务服务器可以将消息流输出节点的消息，自动放入指定的 MQ 队列中。

IBM-MQ 连接器需要在 RCloud 云 EI 服务的管理控制台注册连接器，然后启动该连接器即可。前提是用户已创建了连接器使用的 MQ 队列管理器和队列等。

3.2.8 TongLINK/Q 连接器

TONGLINK/Q 连接器，即可以用作接入类型的连接器，也可以用作接出类型的连接器；用作接入类型的连接器时，可以从指定的 TONGLINK/Q 队列中读取消息，启动消息，用户只需要将消息放入指定的队列中即可；用作接出类型的连接

器时，RCloud 云 EI 服务服务器可以将消息流输出节点的消息，自动放入指定的 TONGLINK/Q 队列中。

TONGLINK/Q 连接器需要在 RCloud 云 EI 服务的管理控制台注册连接器，然后启动该连接器即可。前提是用户已创建了连接器使用的 TONGLINK/Q 队列控制单元（QCU）和队列等。作为接入时的 TONGLINK/Q 连接器配置如下图所示：

注：目前 TONGLINK/Q 连接器支持的 TONGLINK/Q 版本为 7.0.0.1 以上。

3.2.9 Win32Agent 连接器

操作调用流程：

- 第 1 步：CreateAgent
- 第 2 步：Configure 或者 ConfigureFromFile
- 第 3 步：InitAgent
- 第 4 步：进行其他 API 调用
- 第 5 步：DestroyAgent

备注：在产品发布包中，对 Win 32Agent 使用的对象定义（目前只针对 delphi 平台）附带源文件，其中 AgentInfo.pas 定义 Win 32Agent 中 FTP 服务器配置信息、WIN32Agent 本身配置信息等，用户可直接参考使用；Win32AgentApi.pas 如何在开发工程中引用 win32Agent.dll 中的函数。

3.2.9.1 主要对象说明

3.2.9.1.1 FTPInfo:

类型为 TFTPInfo，记录 FTP 服务器相关信息的结构体。

其中 TFTPInfo 的数据结构为：

参数名	类型	描述
ServerIP	array[0..MAX_PATH] of Char	FTP 服务器所在 IP 地址
Port	array[0..MAX_PATH] of Char	FTP 服务器使用的端口号
Username	array[0..MAX_PATH] of Char	FTP 服务器用户登录名称
Password	array[0..MAX_PATH] of Char	FTP 服务器用户登录密码
FTPMode	array[0..MAX_PATH] of Char	FTP 工作方式 (PORT 或者 PASV)
UploadPath	array[0..MAX_PATH] of Char	服务器上供上传文件的文件夹位置
DownloadPath	array[0..MAX_PATH] of Char	FTP 服务器上供下载文件所在位置

3.2.9.1.2 AgentInfo:

类型为 TDEIClientInfo，记录本地客户端信息的结构体

其中 TDEIClientInfo 的数据结构为：

参数名	类型	描述
AgentID	array[0..MAX_PATH] of Char	客户端连接 ID
AgentAlias	array[0..MAX_PATH] of Char	烟站场点的组织代码
SourceNodeId	array[0..MAX_PATH] of Char	对应的 DEI 站点路由名称
SourceNodeIdAlias	array[0..MAX_PATH] of Char	对应的 DEI 站点的别名
MsgPool	array[0..MAX_PATH] of Char	消息池目录
AgentBehavior	TAgentBehavior	发送，接受和处理数据的行为配置
ServerIP	array[0..MAX_PATH] of Char	EI 服务器 IP 地址
ServerPort	array[0..MAX_PATH] of Char	EI 服务器端口
ServerContext	array[0..MAX_PATH] of Char	EI 服务器上下文根

3.2.9.1.3 TAgentBehavior

发送，接受和处理数据的行为配置,数据结构如下：

参数名	类型	描述
MaxTaskCount	Integer	自动接受数据包单次最大数据包个数
ReceiveInterval	DWORD	自动接受数据包的时间间隔单位为毫秒
SyncCallBack	Boolean	自动接受数据包时是否同步调用回调方法
ParallelProcessing	Boolean	并行处理还是串行处理数据包
RetrySendCount	Integer	失败重发的次数
RetrySendInterval	DWORD	失败重发的时间间隔
SendInterval	DWORD	发送池扫描时间间隔
FtpFileLockTimeOut	Integer	下载文件时，对 FTP 目录中的文件锁定超时时间，单位：分钟
HTTPConnectionTimeout	Integer	HTTP 连接和等待响应的超时时间，单位：秒
ext	Array of PChar	备用

3.2.9.1.4 TMsgHead

发送消息时需要设置的传输信息，数据结构为：

参数名	类型	描述
MessageTransportChannel	array[0..MAX_PATH] of Char	服务器间指定传输通道，可选
MessageRelationKeyValue	array[0..MAX_PATH] of Char	唯一标识，可选
MessageInputType	array[0..MAX_PATH] of Char	消息输入类型，可选
MessageSchemaID	array[0..MAX_PATH] of Char	消息业务类型，可选
MessageDestSet	TDest 数组	目的地集合

3.2.9.1.5 TDest

目的地的数据结构：

参数名	类型	描述
-----	----	----

IncludeType	array[0..MAX_PATH] of Char	地址类型：Include 或者 Exclude
ServerExact	array[0..MAX_PATH] of Char	服务器地址匹配模式
ServerType	array[0..MAX_PATH] of Char	服务器类型
ServerValue	array[0..MAX_PATH] of Char	服务器地址
ClientExact	array[0..MAX_PATH] of Char	客户端地址匹配模式，当 exclude 地址时可选
ClientType	array[0..MAX_PATH] of Char	客户端类型，当 exclude 地址时可选
ClientValue	array[0..MAX_PATH] of Char	客户端地址，当 exclude 地址时可选
ProcessorExact	array[0..MAX_PATH] of Char	处理器地址匹配模式，可选
ProcessorType	array[0..MAX_PATH] of Char	处理器类型，可选
ProcessorValue	array[0..MAX_PATH] of Char	处理器地址，可选

3.2.9.1.6 TDataProcessCallBack

回调函数，对从 FTP 服务器下载的数据包进行处理的函数。

TDataProcessCallBack 的函数原型为：

Function (FileName:String;DataPackage: String):Integer;

参数名	类型	描述
FileName	String	为数据包名称
DataPackage	String	解包后的数据

3.2.9.1.7 TRecNotifyCallBack

回调函数，通知客户端数据包的接收状。

TRecNotifyCallBack 的函数原型为：

Function (FileName: Array of String;ExecuteState: TExecuteState):Integer;

参数名	类型	描述
FileName	String	为数据包名称
ExecuteState	TExecuteState	数据包的下载处理过程的状态

3.2.9.1.8 TSandNotifyCallBack

回调函数，通知客户端数据包的发送收状。

TSandNotifyCallBack 的函数原型为：

Function (addition: PChar;executeState: TExecuteState):Integer;

参数名	类型	描述
addition	PChar	为数据关联键值
ExecuteState	TExecuteState	数据包的发送处理过程的状态

3.2.9.1.9 TExecuteState 枚举值：

枚举值	描述
esFound	数据包被发现。
esDownloading	数据包正在下载。
esDownloaded	数据包下载完毕。
esProcessing	数据包正在处理。
esFinished	数据包处理完成。
esError	数据包处理出错（包括发送、下载、处理过程中的错误）。
esSent	数据包发送成功。

3.2.9.2 方法说明

下面列出的是 win32agent 对外提供的接口列表。

3.2.9.2.1 初始化配置

初始化 FTP Agent 环境的相关信息的配置。配置信息包括：ftp 连接需要的相关信息、FTPAgent 的 ID 和 Alias、临时文件夹、EI Route name 、EI 服务器（IP、端口、上下文根）等配置信息。

1. Function CreateAgent():Integer stdcall;

创建 Agent 实例

2. Function Configure(agentInfo: TDEIClientInfo; ftpInfo: TFTPInfo):

integer;stdcall;

用传入的 agentInfo, ftpInfo 设置 Agent 实例的相关配置。

3. Function ConfigureFromFile(fileName: PChar): integer;stdcall;

从配置文件加载配置信息，fileName 为空字符串（"）时将从默认配置

文件（AgenCfg.ini）加载；默认配置文件应该放在与 win32Agent.dll 同一目录下。

4. **Function** InitAgent():Integer;stdcall;

初始化 Agent 的相关内部组件（比如启动发送池等），并连接到 FTP 服务器。

5. **procedure** DestroyAgent(); stdcall;

释放 Agent 实例。

3.2.9.2.2 发送数据包

把业务系统的数据按照一定格式封装成 soap 数据包压缩以 ftp 协议上传到 Ftp 服务器。

1. **Function** SendDataPackage(msgHead:TMsgHead; dataContent:PChar; var dataPackageName:PChar): Integer;stdcall;

根据参数封装成 Soap 格式的压缩包，上传至配置文件指定的 ftp 服务器上。

2. **Function** AsyncSendDataPackage(msgHead:TMsgHead; dataContent:PChar; var dataPackageName:PChar;const SandNotifyCallBack: TSandNotifyCallBack): Integer;stdcall;

异步发送数据包。调用此方法会把数据信息封装后放入发送池在后台发送；此数据包发送成功或失败都会调用 SandNotifyCallBack 回调方法。

3. **Function** SendDataPackageWithAtts(msgHead:TMsgHead; dataContent:PChar; var dataPackageName:Pchar;atts:array of PChar): Integer ;stdcall;

根据消息内容和附件参数封装成压缩包，上传至配置文件指定的 ftp 服务器上。

4. **Function** AsyncSendDataPackageWithAtts(msgHead:TMsgHead; dataContent:PChar;var dataPackageName:PChar;atts:array of Pchar; const SandNotifyCallBack: TSandNotifyCallBack): Integer;stdcall;

异步发送数据包。调用此方法会把数据信息封装后放入发送池在后台发送；此数据包发送成功或失败都会调用 `SandNotifyCallBack` 回调方法。

3.2.9.2.3 接收数据包

把业务系统的数据按照一定格式封装成 `soap` 数据包压缩从 `Ftp` 服务器上以 `ftp` 协议。

1. **Function** `ReceiveDataPackage (dataPackageName: PChar;var dataContent: PChar):Integer; stdcall;`

根据 `dataPackageName` 从 `FTP` 服务器上接收特定数据包，并解包。数据包被下载到本地消息池，函数返回后并不删除。

2. **procedure** `ReceiveCompleted (packageName: PChar); stdcall;`

通知 `Agent` 数据包已经处理完毕，`Agent` 会做些处理完毕后的清理工作。

3. **Function** `GetDataPackageList(var DataPackageList: array of PChar;var count:Integer):Integer; stdcall;`

获取 `FTP` 服务器上指定目录中可下载数据包的列表。

4. **Function** `ReceiveDataPackageWithAtts(dataPackageName: PChar;var dataContent: Pchar;atts:array of PChar):Integer ;stdcall;`

根据 `dataPackageName` 从 `FTP` 服务器上接收特定数据包，并解包。数据包被下载到本地消息池，会将消息内容和附件路径传递给调用方，函数返回后并不删除，需要用户手动删除附件临时文件和所在文件夹

3.2.9.2.4 自动接收数据

可以根据用户的设置，每隔一定时间查看 `FTP` 服务器上是否有可下载的数据包；若存在，即下载到业务系统中

1. **Function** `AutoReceiveStart(interval: Integer; DataProcessCallBack:`

`TDataProcessCallBack; notifyCallBack: TRecNotifyCallBack):Integer; stdcall;`

定时查看 `FTP` 服务器指定位置是否存在可下载数据包，若存在，及下载

到本地，然后调用回调函数对下载的数据包进行处理，并通知客户端接收状态。

2. **Function** AutoRecevieStop ():Integer; **stdcall**;

停止自动接收数据包。

3. **Function** AutoReceviePause ():Integer; **stdcall**;

暂停自动接收数据包。

4. **Function** AutoRecevieContinue ():Integer; **stdcall**;

继续自动接收数据包。

5. **Function** GetDataPackageState (var dataPackagesState: array of TDataPackagesState ; var count:Integer):Integer; **stdcall**;

获取自动接收数据包,所有数据包的当前状态。

6. **Function** AutoReceiveStartWithAtts(interval: Integer; DataProcessCallBack: TDataProcessCallBackWithAtts; nodtifyCallBack: TRecNotifyCallBack):Integer; **stdcall**;

定时查看 FTP 服务器指定位置是否存在可下载数据包，若存在，及下载到本地，然后调用回调函数对下载的数据包进行处理，并通知客户端接收状态。

3.2.9.2.5 网络连接状态获取

获取 FTP 客户端与服务器的连接状态

1. **Function** GetConnectionState():Integer; **stdcall**;

获取 FTP 客户端与服务器的连接状态。返回 1：连接；返回-1：未连接。

3.2.9.2.6 数据打包压缩导出

把业务系统的数据按照一定格式封装成 soap 数据包并压缩

1. **Function** ExportDataPackage(msgHead:TMsgHead;dataContent:PChar;ExportPath: PChar):Integer;**stdcall**;

将导出数据封装为 Soap 格式的压缩包，导出到指定位置。

2. **Function** ExportDataPackageWithAtts(msgHead:TMsgHead;dataContent:PChar;atts:array of PChar;ExportPath: PChar):Integer; **stdcall**;

将导出数据封装为 Soap 格式的压缩包，导出到指定位置。

3.2.9.2.7 数据包的导入

将从业务系统中导出的数据包导入到当前系统中

1. **Function** ImportDataToRemote(fileName :PChar;AAppend: boolean):Integer; **stdcall**;

将导入包导入到本地的上传缓存中。

2. **Function** ImportDataPackageToLocal(fileName:PChar;var dataContent:PChar):Integer; **stdcall**;

将导入包还原为业务系统中的数据，导入到当前业务系统中。

3. **Function** ImportDataPackageToLocalWithAtts(fileName:PChar;var dataContent:Pchar;atts:array of PChar):Integer; **stdcall**;

将导入包还原为业务系统中的数据，导入到当前业务系统中。

3.2.9.2.8 调用 RCloud 云 EI 服务服务器上的代理服务

通过调用 EI 服务器上封装好的代理服务实现同步消息交互。

1. 代理服务封装的接口：

我们提供了代理服务的描述文件：SyncDataTransProxyService.wsdl，其中：

代理服务的名称：	SyncDataTransProxyService		
命名空间：	http://win32.syncTrans		
方法名：	syncTrans		
输入参数（一个）：	名称：dataContent，	类型：string	
输出参数：	名称：result，	类型：string	

在设计工具中采用“指定服务接口”的方式设计代理服务消息流，具体步骤

参照《使用 wsdl 生成代理服务消息流》。

2. 函数原型

Function SendDataPackageOverWS(msgHead:TMsgHead;
dataContent:PChar; var ret:PChar; atts:array of PChar): Integer;stdcall;

其中，ret 为调用代理服务的返回值；msgHead 与 atts 作为保留字段，暂不实现对附件的处理。返回业务服务处理的结果。

3.2.9.2.9 HTTP 方式传输数据

Win32agent 向 RCloud 云 EI 服务服务器传输消息时，使用 HTTP 协议。

Function SendDataPackageOverHTTP(msgHead:TMsgHead;
dataContent:PChar; atts:array of PChar): Integer;stdcall;

目前暂不实现对附件的处理。返回 HTTP 访问的状态码，200 为成功，-1 为在使用方法的过程中出现异常。

3.2.9.2.10 调用 RCloud 云 EI 服务服务器上的代理服务通用接口

函数原型：

Function SendDataPackageOverWSs(wsInfo:TWebServiceInfo;
msgHead:TMsgHead; dataContent:array of PChar; var ret:PChar; atts:TPChar):
Integer;stdcall

函数说明：

调用 EI 服务器上封装好的代理服务。返回业务服务处理后的结果。

参数说明：

wsInfo 是 TWebServiceInfo 结构体中描述的 WebService 的信息

msgHead 是消息头的信息，作为保留字段，暂时不用

dataContent 是 WebService 需要的一些参数值的数组

ret 中存放调用 WebService 后返回的信息

atts 是消息附件列表，作为保留字段，暂时不用

3.2.9.3 代码实例

3.2.9.3.1 初始化

//先创建，再配置，然后初始化。按下列顺序调用：

```
CreateAgent();  
ConfigureFromFile("");  
InitAgent();
```

3.2.9.3.2 数据上传

```
var  
    DataPckName: PChar;  
begin  
    DataPckName:= StrAlloc(MAX_PATH);  
    MessageHead:= self.GetMsgHead;  
    try  
        ret:= SendDataPackage(MessageHead, pchar(msgBody), DataPckName);  
    if ret=1 then      //成功调用  
        //dosomething  
    finally  
        StrDispose(DataPckName);  
    end;  
end;
```

3.2.9.3.3 上传带附件的数据

```
procedure TfrmMain.btnSendWithAttsClick(Sender: TObject);  
var  
    MessageHead: TMsgHead;  
    i: Integer;  
    ret: Integer;  
    DataPckName: Pchar;  
    atts:array[0..2] of PChar;    //附件路径数组  
begin  
    DataPckName:= StrAlloc(MAX_PATH);  
    MessageHead:= self.GetMsgHead;
```

```

atts[0]:=PChar(self.edt_att1.Text);
atts[1]:=PChar(self.edt_att2.Text);
atts[2]:=PChar(self.edt_att3.Text);

Logger.Debug('atts count:'+IntToStr(High(atts)));
Logger.Debug('atts 0:'+atts[0]);

try
    ret:=SendDataPackageWithAtts(MessageHead,
pchar(Trim(meoMsgBody.Lines.Text)),atts, DataPckName);

    if ret<>1 then
        Application.MessageBox(PChar(DataPckName + '发送失败！ '
+ 'Code: ' + IntToStr(ret)), 'MessageSender',
        MB_OK + MB_ICONINFORMATION)
    else
        Application.MessageBox(PChar(DataPckName + '已发送！ '),
        'MessageSender', MB_OK + MB_ICONINFORMATION);
finally
    MessageHead.MessageDestSet := nil;
    StrDispose(DataPckName);
end;
end;

```

3.2.9.3.4 异步发送带附件的数据

```

procedure TfrmMain.RzBitAysncSendWithAttClick(Sender: TObject);
var
    MessageHead: TMsgHead;
    i,count: Integer;
    ret: Integer;
    DataPckName: Pchar;
    atts:array[0..2] of PChar;    //附件路径数组
begin
    DataPckName:= StrAlloc(MAX_PATH);
    MessageHead:= self.GetMsgHead;
    atts[0]:=PChar(self.edt_att1.Text);
    atts[1]:=PChar(self.edt_att2.Text);
    atts[2]:=PChar(self.edt_att3.Text);

    Logger.Debug('atts count:'+IntToStr(High(atts)));

```

```

    Logger.Debug('atts 0:'+atts[0]);

    count:= self.edt1.IntValue;
    if count=0 then count:=1;
    try
        for i:=0 to count-1 do
            begin
                ret:=AsyncSendDataPackageWithAtts(MessageHead,
                    pchar(Trim(meoMsgBody.Lines.Text)),atts,
                    DataPckName,Notify);
                if ret<>1 then
                    Application.MessageBox(PChar(DataPckName
                        + '发送失败！' + 'Code: ' + IntToStr(ret)),
                        'MessageSender', MB_OK + MB_ICONINFORMATION)
                end;

                if count=1 then
                    if ret=1 then
                        Application.MessageBox(PChar(DataPckName
                            + '已发送！'), 'MessageSender', MB_OK
                                + MB_ICONINFORMATION);
                    finally
                        MessageHead.MessageDestSet := nil;
                        StrDispose(DataPckName);
                    end;
                end;
            end;
        end;
    end;
end;

```

3.2.9.3.5 接收带附件的数据

```

procedure TfrmMain.RzBtnReceiveDataPkgWithAttClick(Sender: TObject);
var
    frm: TfrmInputBox;
    data: PChar;
    frm2: TfrmShowBox;
    ret: Integer;
    atts: array of PChar;
    i, count: integer;
    content: String;
begin
    setlength(atts, 10);
    for i:=0 to (Length(atts)-1) do

```

```

begin
    atts[i]:=StrAlloc(MAX_PATH);
    StrCopy(atts[i],""); //内存初始化为 ""
end;

frm:= TfrmInputBox.Create(self);
if frm.ShowModal = MROK then
begin
    data:= StrAlloc(1024*1024*2);
    try
        ret:=ReceiveDataPackageWithAtts(
            PChar(frm.PackageName),data,atts);

        frm2:= TfrmShowBox.Create(self);
        if ret=1 then
        begin
            count:=0;
            content:=string(data)+#13#10+'附件： ';
            for i:=0 to (Length(atts)-1) do
            begin
                if Strlen(atts[i])=0 then
                    continue
                else
                begin
                    count:=count+1;
                    content:=content+#13#10+atts[i];
                end;
            end;

            if count=0 then
                content:=content+'没有.';

            frm2.Text:= content;

        end
        else
        begin
            frm2.Text:= '接收失败， 请查看日志！ ';
        end;
        frm2.ShowModal;
    finally
        frm2.Free;
    end;
end;

```

```
StrDispose(data);  
for i:=0 to (Length(atts)-1) do  
begin  
    if (atts[i]<>nil) and (atts[i]<>'') then  
    begin  
        StrDispose(atts[i]);  
    end;  
end;  
end;  
  
frm.Free;  
end;
```

3.2.9.3.6 自动接收带附件的数据

```
procedure TfrmMain.RzBntAutoRecWithAttsClick(Sender: TObject);  
var  
    process: TDataProcessCallBackWithAtts;  
    nodyfy: TRecNotifyCallBack;  
begin  
    process:= ProcessDataWithAtts;  
    nodyfy:= Notify ;  
    AutoReceiveStartWithAtts(-1,process,nodyfy);  
end;
```

3.2.9.3.7 自动接收带附件的数据的回调函数

```
function ProcessDataWithAtts(dataPackageName, dataPackage:  
    PChar;atts:array of PChar): Integer;  
var  
    Data: PChar;  
    pckname: PChar;  
    filePath: string;  
    dataList: TStrings;  
    agentInfo: TDEIClientInfo;  
    i:Integer;  
  
begin  
    Logger.Info('ProcessDataWithAtts is invoked.');
```



```

dataList:= TStringList.Create;
dataList.Text:= StrPas(dataPackage);
filePath:= ExtractFileName(StrPas(dataPackageName)) + '.xml';
agentInfo:= GetAgentInfo();
dataList.SaveToFile(agentInfo.MsgPool + filePath);

Loger.Info('Data Content save to file:'+filePath);
if Length(atts)=0 then
begin
    Loger.Info('消息未带附件. ');
end
else
begin
    Loger.Info('附件个数: '+intToStr(Length(atts)));
    for i:=0 to Length(atts)-1 do
    begin
        if (atts[i]<> nil) and (atts[i]<>'') then
            Loger.Info('附件 '+IntToStr(i)+' : '+atts[i]);
        end;
    end;
finally
    dataList.Free;
end;
Loger.Info('ProcessDataWithAtts run over. ');

result:=1;
end;

```

3.2.9.3.8 导出带附件的数据

```

procedure TfrmMain.RzBitSaveWithAttClick(Sender: TObject);
var
    MessageHead: TMsgHead;
    i: Integer;
    ret: Integer;
    DataPckName: Pchar;
    dir: string;
    atts:array[0..2] of PChar;    //附件路径数组
begin
    atts[0]:=PChar(self.edt_att1.Text);
    atts[1]:=PChar(self.edt_att2.Text);

```

```
atts[2]:=PChar(self.edt_att3.Text);

Logger.Debug('atts count:'+IntToStr(High(atts)));
Logger.Debug('atts 0:'+atts[0]);

MessageHead:= self.GetMsgHead;
if self.SelectFolderDialog(self.Handle,
    '选择保存位置',"dir) then
begin
    ret:= ExportDataPackageWithAtts(MessageHead,pchar(Trim(
        meoMsgBody.Lines.Text)),atts,Pchar(dir+'\'));
    if ret <> 1 then
        Application.MessageBox(PChar('失败！ '),
            'MessageSender', MB_OK + MB_ICONINFORMATION)
    end;
end;
```

3.2.9.3.9 导入带附件的数据到本地

```
procedure TfrmMain.RzBtnImport2LocalWithAttsClick(Sender: TObject);
var
    data: PChar;
    ret: integer;
    frm: TfrmShowBox;
    atts:array of PChar;
    i,count:integer;
    content:String;
begin

    setlength(atts,10);
    for i:=0 to (Length(atts)-1) do
        begin
            atts[i]:=StrAlloc(MAX_PATH);
            StrCopy(atts[i],""); //内存初始化为 "
        end;

    data:= StrAlloc(1024*1024*2); //返回的是 UTF-8 编码的字符串
    try
        if self.OpenDialog1.Execute then
            begin
```

```

ret:= ImportDataPackageToLocalWithAtts(
    Pchar(self.OpenDialog1.FileName),data,atts);
if ret <> 1 then
    Application.MessageBox(PChar('失败！'), '提示',
        MB_OK + MB_ICONINFORMATION)
else
begin
    count:=0;
    content:=string(data)+#13#10+'附件：';
    for i:=0 to (Length(atts)-1) do
    begin

        if Strlen(atts[i])=0 then
            continue
        else
            begin
                count:=count+1;
                content:=content+#13#10+atts[i];
            end;
        end;

        if count=0 then
            content:=content+'没有.';
        frm:= TfrmShowBox.Create(self);
        frm.Text:=content;
        frm.ShowModal;
        frm.Free;
    end;
end;
finally
    StrDispose(data);
end;
end;

```

3.2.9.3.10 数据下载

```

var
    data: PChar;
begin
    data:= StrAlloc(1024*1024*2);    //分配空间
    try

```

```
ret:= ReceiveDataPackage(PChar(PackageName),data);  
if ret=1 then //调用成功  
begin  
    //do things  
end;  
finally  
    StrDispose(data); //释放空间  
end;  
end;
```

3.2.9.3.11 获取下载列表

```
var  
    list: array[0..9999] of PChar;  
    strList: TStringList;  
    i,count,ret: integer;  
begin  
    for i:=0 to High(list) do  
        begin  
            list[i]:= StrAlloc(MAX_PATH);  
        end;  
    ret:= GetDataPackageList(list,count);  
  
    if ret=1 then  
        begin  
            strList:= TStringList.Create;  
            for i:=0 to count-1 do  
                begin  
                    strList.Add(Strpas(list[i]));  
                end;  
            //do things  
            strList.Free;  
        end;  
  
        for i:=0 to High(list) do  
            begin  
                StrDispose(list[i]);  
            end;  
        end;  
    end;
```

3.2.9.3.12 文件下载状态获取

```
var
  List: array of TDataPackagesState;
  strList: TStrings;
  i,count: integer;
begin
  SetLength(list,MaxListSize);

  if GetDataPackageState(List,count)=1 then
    begin
      strList:= TStringlist.Create;
      for i:=0 to count-1 do
        begin
          //do things
        end;
      //do things
    end;
    list:= nil;
end;
```

3.2.9.3.13 调用代理服务

```
var
  ret : PChar;
  dataContent : PChar;
  msgHead : TMsgHead;
  atts: array of PChar;
  msgBody:string;
begin
  msgBody := Trim(meoMsgBody.Lines.Text);
  dataContent := @msgBody[1];
  ret:= StrAlloc(1024);

  SendDataPackageOverWS(msgHead, dataContent, ret,atts);
  meoMsgBody.Lines.Text := ret;
  ShowMessage(ret) ;
  StrDispose(ret);
end;
```

3.2.9.3.14 HTTP 方式传输消息

```
var
    dataContent : PChar;
    msgHead : TMsgHead;
    atts: array of PChar;
    msgBody:string;
begin
    ConfigureFromFile('');

    msgHead:=self.GetMsgHead;;
    msgBody := Trim(meoMsgBody.Lines.Text);
    dataContent := @msgBody[1];
    result:=SendDataPackageOverHTTP(msgHead, dataContent,atts);
end;
```

3.2.9.3.15 调用代理服务通用接口

```
var
    ret : PChar;
    dataContent : array[0..1] of PChar;
    msgHead : TMsgHead;
    atts: array of PChar;
    msgBody:string;
    wsInfo:TWebServiceInfo;
    intret:Integer;
begin

    msgHead:=self.GetMsgHead;;
    msgBody := Trim(meoMsgBody.Lines.Text);
    dataContent[0] := 'test1';
    dataContent[1] := 'test2';

    ret:= StrAlloc(1024*3);
    wsInfo.URL := 'http://192.9.107.18:7001/rode/services/errfiledownload';
    wsInfo.Namespace := 'http://webservice.comms.sitrpbp.icss.com';
    wsInfo.MethodName := 'errfiledownload';
    wsInfo.prefix := 'tns';
```

```
wsInfo.Qualified := 1;
SetLength(wsInfo.ParamNames, 2);
wsInfo.ParamNames[0] := 'arg0';
wsInfo.ParamNames[1] := 'arg1';
wsInfo.returnValue := 'errfiledownloadReturn';
intret := SendDataPackageOverWSs(wsInfo, msgHead, dataContent, ret,atts);
if intret=1 then
begin
    meoMsgBody.Lines.Text := ret;
    StrDispose(ret);
end
else
    ShowMessage('webservice-ÃËÊ$°Ü£¡') ;

end;
```

3.2.10 数据库触发器

3.2.10.1 初始配置

参见《ResourceOne4.5 Exchange-Integration 产品用户使用手册》文档中相关章节，完成前置的配置工作。

3.2.10.2 创建过程

应用可以不通过 SDK API，通过数据库触发器调用 JAVA UDF 的方式直接向交换服务器发送消息。

数据库触发器方式需要创建一个函数如下。

```
SENDTOR1DE(CLIENTSHORTNAME varchar(150), RKEY varchar(50), DATA
varchar(1000))
```

函数参数说明：

1、CLIENTSHORTNAME：应用客户端简称，对数据库每个触发器都有一个唯一的应用客户端 ID；

2、RKEY：关联键的值，同 R1DEClient 中的关联键的值，目前可以放空字符

串；

3、DATA: 要发送的数据，data 要按照一定的格式拼写：“数据名称=数据值#”，可以连接任意条数据，例如：username=myname#age=27。

当表发生关心的变化后，数据库触发器调用函数 sendToR1DE，即可将消息发送。

3.2.10.3 代码示例

建立触发器：

```
CREATE TRIGGER DB2ADMIN.TESTDETRIGER
AFTER
UPDATE OF
NAME
ON DB2ADMIN.TEST
REFERENCING
OLD AS oldrow
NEW AS newrow
FOR EACH ROW
MODE DB2SQL
values(sendToR1DE('CRM_001','','username=myname#age=27'));
```

3.2.10.4 消息格式

触发器触发后，RCloud 云 EI 服务服务器会包装数据库触发器发送的数据，比如上面所描述的数据：username=myname#age=27。那么，会包装成：

```
<DeTrigger>
  <username>myname</username>
  <age>27</age>
</DeTrigger>
```

在使用消息流程 RCloud 云 EI 服务设计工具设计流程时，SQL 扩展节点的 SQL 脚本内可以使用带“标签”的 xpath，在运行时，引擎会使用 xpath 查询消息，取出相应的值回填到 SQL 脚本中，从而执行该脚本，比如如下 SQL 脚本：

```
INSERT INTO YOUR_TABLE (COLNAME) VALUES(
'<demessage-name-tag>/DeTrigger/username</demessage-name-tag>');
```

注意如上所示黑体字所示的 xpath: **/DeTrigger/username**，因为 DE 服务器

端包装消息后使用“DeTrigger”作为 xml 的根元素，所以要查找“username”的值就使用“/DeTrigger/username”。

另外，在 RCloud 云 EI 服务设计工具设计个人工作台节点时，也可以使用带标签的 xpath，比如：

个人工作台的标题为：

今日卷烟销售数量：

<demessage-name-tag>/DeTrigger/username</demessage-name-tag>万条。

请注意：上面所说的 XPATH 必须是全路径名，比如 XPATH：“/DeTrigger/username”，虽然对于 XPATH 的语法来说可以写“//username”，也可以搜索到 username 元素的值，但是，DE 引擎不能够处理“// username”这样的 XPATH。

3.3 别名路由

别名路由就是为 RCloud 云 EI 服务服务器节点和 RCloud 云 EI 服务所管理的应用提供多个名称进行标识并可以根据不同的标识进行路由转发和处理。在发送端，用户可以为同一目的地节点或应用指定不同名称作为标识，并根据该标识路由到达目的地。

目前除了 IBM-MQ 连接器接入、FTP、TongLINK/Q 连接器和文件连接器接入不支持别名路由外，其它连接器均支持别名的方式进行路由。

使用别名的方式进行路由，可以直接指定消息接收的目的地信息，不需要绑定消息流。

3.4 RCloud 云 EI 服务服务器间传输方式

RCloud 云 EI 服务服务器间支持多种传输方式，如 IBM WebSphere MQ、HTTP、TongLINK/Q，当 RCloud 云 EI 服务服务器间传输方式为 IBM -MQ 或 TongLINK/Q 时，可配置服务器间多通道的传输。

用户可以规范行业中不同业务的传输使用不同的通道，避免了不同业务数据传输时的相互影响，减少因一种业务堵塞整个传输通道，从而提高了传输的可靠

性和并发处理能力。

使用指定通道进行传输的时候，要求在发送方和接收方的整个传输路径中每两个相邻的 RCloud 云 EI 服务服务器之间配置有该通道标识的传输通道，否则消息将无法正确传输。

3.4.1 传输方式中多 MQ 通道/TongLINKQ 连接的使用

3.4.1.1 DE Agent 通过开发时在程序中使用相应 API 指定

(1)、普通 DE Agent

代码示例

```
R1DEClient client = new R1DEClient();
MessageHeader header = new MessageFlowHeader();
//在发送的消息头中指定传输链路和通道标识
header.setTransportChannel("链路标识.通道标识");
.....
client.setMessageHeader(header);
.....
```

3.4.1.2 SOAP/WS 连接器

在 SOAP/WS 连接器中指定传输方式或 MQ 通道可以根据需要通过调用 WsConnectionService 的下列方法实现：

1、指定传输通道的服务调用，包括基本服务调用参数和传输链路标识

```
public boolean receiveReqMsgWithChannel(String keyVal, String
sourceClientId, String messageBody, String schemaId, String msgTransportChannel)
```

2、指定目的地和传输通道的服务调用，包括基本服务调用参数、传输链路标识和目的地信息

```
public boolean receiveReqMsgWithDestChannel(String keyVal, String
sourceClientId, String messageBody, String schemaId, String destNodeId, String
destClientId, String msgTransportChannel, String messageAdaptId)
```

3、指定目的地别名和传输通道的服务调用，包括基本服务调用参数、传输

链路标识和目的地别名信息

```
public boolean receiveReqMsgWithDestAliasChannel(String keyValue,String
sourceClientID,String messageBody,String schemaId,String destNodeAlias,String
destClientAlias, String msgTransportChannel,String messageAdaptId)
```

3.4.1.3 HTTP 连接器

HTTP 连接器需指定传输方式或 MQ 通道/TongLINKQ 连接时，需要传递 messageTransportChannel 参数来指定。

3.4.1.4 其它连接器通过配置指定

1、文件连接器

使用文件连接器接入时，需要在连接器中配置“服务器间传输链路”参数来指定传输链路或 MQ 通道/TongLINKQ 连接，供消息发送时使用。如下图所示：



2、兼容 Infobus 的 DE Agent

由于 RCloud 云 EI 服务兼容 InfoBus 的 SDK 中没有提供指定传输链路或 MQ 通道的方法，因此，需要手动修改配置文件，在 RodeClientConfig.xml 中注册 TransportChannel 标识，如下图所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<GlobalRodeClient>
  <Client Name="agent1" Type="web">
    <UUID>4089ebb7-1993cb8a-0119-93cd6645-0003</UUID>
    <CnName>agent.in</CnName>
    <Token>BsUSaYbYrROUWmxhXdBvyVtsF59CCYQ9V44yXU0f3111BYIp3e4
ifg69RE0Ht5iozwZuMG4F4LUWD3ZLjpTeoB6Kr9wV+kuEyjvOn3zmAzLWX
NXMjBZTj5d5nLUwHDM4QytFUIIBk9PoKG+m+v+fw4ZLz2Y6WDXtJ20QX9W
OGwk=</Token>
    <status>2</status>
    <ServerRouteName>server1</ServerRouteName>
    <ServerUrl>http://192.168.0.1:9080/rode/DETServlet</ServerUrl>
    <MQRcvQueue>1</MQRcvQueue>
    <TransportType>1</TransportType>
    <TransportChannel>mq16_28.cl</TransportChannel>
  </Client>
</GlobalRodeClient>
```

3、IBM-MQ 连接器

同文件连接器接入一样，需要在连接器中配置“服务器间传输链路”参数来指定传输链路或 MQ 通道/TongLINKQ 连接。如下图所示：

配置连接器：IBM-MQ连接器

基本信息 MQ队列配置 传输配置

目的地服务器路由名称：

目的客户端ID：

消息处理器ID：

服务器间传输链路： OK

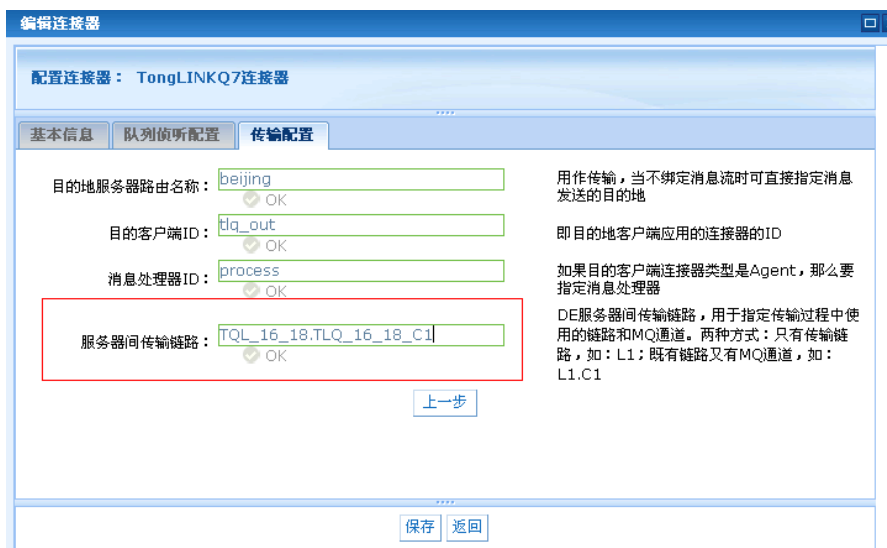
上一步

添加 返回

用作传输，当不绑定消息流时可直接指定消息发送的目的地
即目的地客户端应用的连接器的ID
如果目的客户端连接器类型是Agent，那么要指定消息处理器
DE服务器间传输链路，用于指定传输过程中使用的链路和MQ通道。两种方式：只有传输链路，如：L1；既有链路又有MQ通道，如：L1.C1

4、TongLINK/Q 连接器

同 IBM-MQ 连接器接入一样，需要在连接器中配置“服务器间传输链路”参数来指定传输链路或 MQ 通道/TongLINKQ 连接。如下图所示：



3.5 定时任务

定时任务是在 RCloud 云 EI 服务服务器端通过定时任务调度器定期执行的一种机制。用户把自己的实现类放入 RCloud 云 EI 服务服务器应用的 classpath 下并创建定时任务，设置定时任务执行规则即可。

3.5.1 接口

`com.icss.ro.de.server.schedule.AbstractJob` 是定时任务抽象类，用户自己编写的定时任务必须继承这个类，实现抽象方法，通过 RCloud 云 EI 服务服务器端的定时任务调度器定期执行。

3.5.2 方法说明

任务执行方法

```
public abstract void executeJob() throws TimeJobException
```

3.5.3 代码示例

普通任务的写法：

```
public class MyJob extends AbstractJob {
```

```

/* 实现了 AbstractJob 中的抽象方法*/
public void executeJob() throws TimeJobException {
    Debug.debug("定时任务执行，当前时间： " + new Date());
}
}

```

3.5.4 启动消息流

3.5.4.1 说明

使用定时任务启动消息流。通常用于根据业务需要，经过一定的时间间隔，统计当前的业务数据总量或者判断某些业务数据是否超过阈值，从而启动相应的统计上报消息流程或者报警消息流程。

3.5.4.2 代码示例

```

public class StartProcessJob extends AbstractJob {
    /* (non-Javadoc)
     * @see com.icss.ro.de.server.schedule
     * .AbstractJob#executeJob()
     */
    public StartProcessJob(){
    }

    public void executeJob() throws TimeJobException {
        //获得业务 xml 数据,可能是业务人员写的方法
        String xmldata= getXmlMessage();
        //获得关联键值，可能是业务人员写的方法
        String relationKeyValue = getRelationKeyValue();
        //接入消息流的连接器 ID，可能是业务人员写的方法
        String sourceClientId = getSourceClientId();
        try {
            startMessageFlow(xmldata,
                relationKeyValue, sourceClientId);
        } catch (Exception e) {
            throw new TimeJobException(
                "my exception detail", e);
        }
    }
}

```

```

    }
    /**
     * 发送消息
     */
    private void startMessageFlow (String xmldata,
                                   String relationKeyValue,String sourceClientID)
        throws DEDMessageFormatException
        , DETransportException {
        //获得本地节点路由名称
        DEDirectoryManager serverManager =
            DEDirectoryManager.getInstance();
        String localRouteName =
            serverManager.getLocalNode().getRouteName();

        MessageWrapper message = new MessageWrapper ();
        message.setMessageSourceClientId(sourceClientID);
        message.setMessageSourceNodeId(localRouteName);
        message.setMessageDestinationNodeId (localRouteName);
        message.setMessageRelationKeyValue(
            relationKeyValue);
        message.setMessageID(new UUIDHex().toString());
        message.setMessageStringBody(xmldata);
        //消息类型：请求/反馈
        message.setMessageDirection(DEServerConstants
            .MESSAGE_EXCHANGE_DIRECTION_REQ);
        DETMsgDispatcher.dispatch(message);
    }
}

```

3.6 消息流 Java 扩展节点

Java 扩展节点是用户对消息流功能的扩展。如果用户需要在消息流中增加自己的业务处理逻辑，可以在消息流中增加该种类型的节点，并为该节点开发业务逻辑代码。将这些业务逻辑代码编译成 jar 包，再把该 jar 包放到 EI 应用服务器的 lib 目录下。

3.6.1 接口

com.icss.ro.de.server.flow.engine.impl.WfAbstractJavaExtensionApplication 是消

息流 Java 扩展的抽象类，用户自己编写的 Java 扩展必须继承这个类，实现抽象方法，通过 RCloud 云 EI 服务服务器的消息流引擎执行。

3.6.2 方法说明

- 1、任务执行方法，需要用户实现（返回值 null）

public abstract Map run(Map context) **throws** JavaExtensionException

- 2、是否需要 RCloud 云 EI 服务服务器保存消息数据 true or false，需要用户实现

public boolean needSave()

- 3、获得当前消息流程实例中的消息对象数组

protected MessageWrapper[] getMessages(Map context) **throws**

JavaExtensionException

- 4、获得当前消息流程实例中的消息 ID 列表

protected List getMessageIds (Map context) **throws** JavaExtensionException

- 5、设置当前消息流程实例中的消息对象数组

protected void setMessages(MessageWrapper[] messages)

用户对消息数组中的消息数据进行操作后，不需要人工干预保存消息数据，引擎会自动根据方法 needSave()的返回值判断是否保存。

3.6.3 代码示例

消息流 Java 扩展的写法：

```
public class JavaExtensionSample
    extends WfAbstractJavaExtensionApplication {
    public Map run(Map context)
        throws JavaExtensionException{
        //获得消息对象
        MessageWrapper[] messages = getMessages(context);
        String xmldata = messages[0].getMessageStringBody();
        //.....对 xmldata 处理
        return null;//请返回 null
    }
}
```



```
public boolean needSave(){
    return false;//如果需要保存则返回 true
}
}
```

3.7 消息流 SQL 扩展节点

消息流中的 SQL 扩展节点，可以自动执行用户填写的 SQL 语句，而且 SQL 语句中使用的值，可以是当前消息流中消息内容中的值，用 XPATH 方式指定。

例如 SQL 语句为：

```
INSERT INTO YOUR_TABLE (COLNAME)VALUES
('<demessage-name-tag>/root/customername</demessage-name-tag>');
```

标签<demessage-name-tag>和</demessage-name-tag>之间为要从消息内容中取得值的 XPATH，该 XPATH 必须是绝对路径的，如果 XPATH 可以查询到多个值，那么只取第一个值。

3.8 消息流中 Web 服务调用

消息流程中的【服务调用】节点，可以同步调用用户配置的外部 Web 服务，如果被调用的服务有响应消息，那么消息流引擎会根据节点的响应处理设置自动将响应消息的内容插入或替换到 XPATH 指定的源消息位置，消息流向下游流转时，会使用更新后的消息内容。

3.8.1 Web Service 配置

用户可以选择通过静态或动态的方式选择 WEB 服务的地址，静态寻址就是直接在界面中输入 WEB 服务的访问地址；动态寻址是指由 RCloud 云 EI 服务服务器运行时动态地从指定的统一资源库中取得目标服务的访问地址。

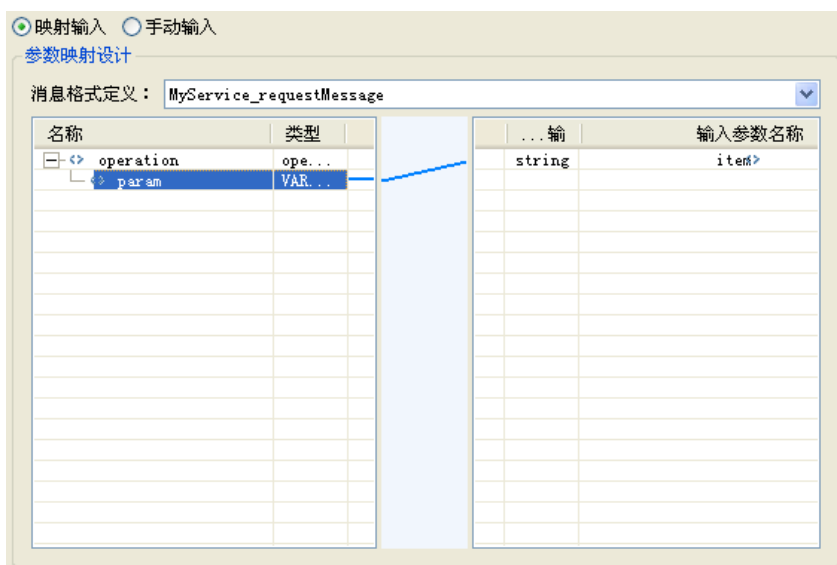
WEB 服务的配置还需要用户提供该服务的 WSDL 文件，然后由设计工具对服务的 WSDL 文件进行解析，获取服务的访问地址、操作名、命名空间、请求及响应参数等信息。可以用户通过配置统一资源库浏览远程或选择本地已注册的

WSDL 资源两种方式来指定 WEB 服务的 WSDL 文件。

3.8.2 服务参数映射

服务的形式参数是指访问当前服务操作的输入参数。实际调用服务时，用户需要将所有的形式参数值封装在一个 XML 消息中，并且为每个形式参数指定一个访问的 XPATH。服务在响应输入时会根据指定的 XPATH 从消息体中获取参数值并进行相应的类型转换。关于 XPATH 的格式请参照 W3C 的 XML Path Language (XPath) Version 1.0（1999-11-16）规范。

形式参数的设置有两种方式：映射输入和手动输入。映射输入的方式跟消息映射的操作是类似的，即选定输入消息的 schema，将输入的形式参数与消息中指定位置的值一一进行映射关联；手动输入方式就是为每个形式参数填写实际参数所对应应在输入消息中的 XPATH。如下图所示。



映射输入参数映射

3.9 消息流中长周期 Web 服务调用节点

长周期 Web 服务调用节点，也是对外部用户提供的 Web 服务的一种访问方式。该 WEB 服务配置、参数映射、响应处理和“Web 服务调用”相同，可以参考 3.7 小节的内容。“长周期 Web 服务调用功能”和“Web 服务调用”功能的区别是：

1、“长周期 Web 服务调用节点”功能在消息流中对应使用的是【长周期服务调用】节点，“Web 服务调用”使用的是【服务调用】节点

2、“Web 服务调用”功能是同步执行的，也就是 Web 服务会在同一个会话中将响应返回给消息流【服务调用】节点。Web 服务请求会话持续时间受 HTTP 协议的限制，允许的响应返回时间较短。“长周期 Web 服务调用”是一种异步返回结果的方式，【长周期服务调用】节点，调用 Web 服务后，调用会话结束，挂起当前消息流节点，当被调用的 Web 服务方处理完成，回调 RCloud 云 EI 服务提供的接收响应消息的 Web 服务，将反馈返回给挂起的消息流后，消息流引擎会恢复被挂起的环节继续流转。

3.9.1 回调服务

用户的外部服务执行完成后，需要回调 RCloud 云 EI 服务提供的接收反馈消息的服务把反馈消息发送给 RCloud 云 EI 服务服务器。

- 1、服务名：WSConnectionService
- 2、版本：1.0
- 3、命名空间：http://webService.connector.de.ro.icss.com/xsd
- 4、服务方法：

```
public String receiveResponseMessage(String relationKey
    , String messageBody)
```

参数 relationKey: 关联键，传递外部服务接收到的关联键值

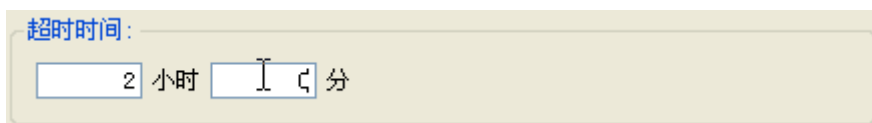
参数 messageBody: 反馈消息内容，该内容会根据消息流中的响应设置对消息进行填充或替换。

- 5、服务访问地址：http://[服务器地址]:[端口]/[上下文]

根]/services/WSConnectionService

3.9.2 超时处理

长周期服务调用节点可以设置反馈消息超时时间，如果超过设置的超时时间，消息流引擎会自动恢复挂起的节点，继续执行，但会将消息流路由到该节点连接的错误处理节点，如果没有异常处理节点，消息流自动结束。超时时间设置如下图所示：



超时时间: 小时 分

长周期服务调用超时时间设置

3.9.3 代码示例

长周期服务调用回调服务代码示例：

```
public String echobaselong(String rk){
    //模拟长周期 WEB 服务回调服务
    Thread t = new Thread(rk){

        public void run(){
            String url =
                "http://192.9.107.18:7001/rode/services/" +
                "WSConnectionService";
            QName qName = new QName(
                "http://webservice.connector.de.ro.icss.com
                /xsd",
                "receiveResponseMessage");
            Object[] _opArgs = new Object[]{this.getName(),
                "<retmsg>It's a return msg for long service " +
                "call.abcd</retmsg>"};
            Class[] _returnTypes = new Class[]{boolean.class};
            try {
                RPCServiceClient serviceClient =
                    new RPCServiceClient();
                Options options = serviceClient.getOptions();
                EndpointReference targetEPR =
                    new EndpointReference(url);
```

```

options.setTo(targetEPR);
options.setAction(
    "urn:receiveResponseMessage");
serviceClient.invokeBlocking(
    qName, _opArgs, _returnTypes);
} catch (AxisFault e) {
    Debug.error(e,e.getMessage(),module);
}
}
};

try {
    t.sleep(new Long(1000*60).longValue());
    t.start();
} catch (InterruptedException e1) {
    Debug.error(e,e.getMessage(),module);
}
return "<msg>ok ok</msg>";
}

```

3.10 消息流错误处理节点

错误处理节点是一种特殊的 Java 扩展节点，由用户扩展实现，用于处理消息流节点中出现的异常情况、业务操作的事务补偿等目的。如下图示例所示：



消息流错误处理节点设计示例

在 RCloud 云 EI 服务设计工具里可以看到，JAVA 扩展节点、SQL 扩展节点、消息映射节点，服务调用节点，长周期服务调用节点，都带有错误处理输出小端点，可以使用迁移线连接到错误处理节点。比如 SQL 扩展节点里 sql 语句的 xpath 写错时、或者消息映射节点映射消息数据出错时。如果多个节点有迁移线连接到

错误处理节点，那么 RCloud 云 EI 服务消息流引擎会包装一个错误块发送到错误处理节点。该错误块包括：

- 1、出错前的消息 ID 列表（用逗号隔开）
- 2、出错时是否修改过该消息
- 3、出错后的消息 ID 列表（用逗号隔开）

如果修改过消息，那么就可以得到出错后的消息 ID 列表；如果没有修改过该消息，那么出错后的消息列表就为空。

使用错误处理节点，需要用户自己写继承自 WfAbstractJavaExtensionApplication 的处理消息的实现类，并且需要在 RCloud 云 EI 服务设计工具设置。

3.10.1 代码示例

```
public class ErrorHandlerJavaExtApp
    extends WfAbstractJavaExtensionApplication {
    public Map run(Map context)
        throws JavaExtensionException {
        //取得错误块
        ErrorBlock errorBlock = (ErrorBlock)
            context.get(WfContextKeys.ERROR_BLOCK);
        //取得发生错误前、还未被修改的消息数据的 ID
        List changeBefMsgIds = errorBlock.getMsgIds();
        //准备通过消息 ID 获得消息体
        MessageWrapper changeBefMsg = null;
        ServiceFactory sf = ServiceFactory.newInstance();
        MessagePersistence mp = sf.getMessagePersistence(
            ServiceConstants.SERVICE_PERSISTENCE_TYPE_DB,
            false);
        for (Iterator it = changeBefMsgIds.iterator();
            it.hasNext();) {
            //原 xml 消息
            changeBefMsg = mp.getMessageWrapper (
                (String)it.next() );
            //TODO 具体应用可能会对原消息有自己的处理方式
            // 比如，使用改变后的新消息恢复数据库等等
            Debug.debug("得到错误块中的原消息:"
                + changeBefMsg.getMessageStringBody());
        }
    }
}
```

```

    }
    //错误描述 ， 具体应用可能会把出错描述输出的一些设备上
    Debug.debug("出错描述: " + errorBlock.getErrorDesc());
    //如果源消息被改变，那么得到改变过的消息
    if(errorBlock.isChanged()){
        // 得到改变后的消息数据
        List newMsgIds = errorBlock.getNewMsgIds();
        // 准备通过 ID 获得改变后的消息
        MessageWrapper newMsg=null;
        for (Iterator it = newMsgIds.iterator();
            it.hasNext();) {
            newMsg = mp.getDEMessageById(
                (String)it.next());
            if(newMsg!=null) {
                //错误描述 ， 具体应用可能会对改变后的新消息有
                //自己的处理方式比如，使用改变后的新消息
                //恢复数据库等等
                Debug.debug("得到改变后的新消息:\n"
                    +newMsg.getMessageStringBody () );
            } else {
                //具体应用可能针对改变后的消息丢失会有自己的处理
                Debug.debug("新消息为空，可能改变后的新消息" +
                    "已经丢失！ ");
            }
        }
    }
    return null;
}
}

```

3.11 消息多目的地分发

消息多目的地分发功能目前只对通过 DEAgent 发送的消息有效。通过 DEAgent 提供的 API，可以设置多个消息发送目的地，也可以按一定规则模糊指定消息发送的目的地。

3.11.1 多目的地地址指定规则

3.11.1.1 明确指定多个地址

确切的指定目的地 RCloud 云 EI 服务服务器路由名称/别名和连接器路由名称/别名，通过调用 `DestinationSet` 提供的方法，指定多个目的地地址（一个完整的地址需要包含服务器路由名/别名、连接器 ID/别名，如果接收端是 `DEAgent`，还需要带上消息处理器 ID/ClassName）。

代码示例：

```
.....
// 服务器指定方式
String serverAddressType = DestinationSet.ADDRESS_TYPE_ROUTENAME;
// 服务器值:路由名称
String serverAddressValue = "Leshan";
// 是否精确指定
boolean serverAddressExact = true;

// 连接器指定方式
String clientAddressType = "ClientID";
// 连接器值: 连接器 id
String clientAddressValue = "tlam";
// 是否精确指定
boolean clientAddressExact = true;

// 消息处理器指定方式
String processorValueType = "ID";
// 消息处理器值:消息处理器 id
String processorValue = "tlamProecessor";
// 是否精确指定
boolean processorValueExact = true;

DestinationSet dest = new DestinationSet();
dest.includeAddress(serverAddressType, serverAddressValue,
    serverAddressExact,clientAddressType,clientAddressValue,
    clientAddressExact,processorValueType,processorValue,
    processorValueExact);

// 添加第二个地址
```

```
dest.includeAddress(DestinationSet.ADDRESS_TYPE_ROUTENAME,"Chengdu",true,
    "ClientID","tlpm",true,
    "ID","tlpmProcessor",true);
```

3.11.1.2 模糊指定目的地

1. 服务器路由名称模糊指定：

在 RCloud 云 EI 服务总线中的 EI 服务器，其路由名称有层级关系，所以可以使用如下方式表示：

地址表达式	含义
<code>[%RouteName%]%R%</code>	指 EI 路由拓扑图中，从[%RouteName%]EI 服务器向根节点方向 %R% 级的所有 EI 服务器，不包括[%RouteName%] EI 服务器； %R% 是正整数或者 N，当为 N 时，表示从[%RouteName%]向根节点方向的所有 EI 服务器； 表达式[%RouteName%]表示 EI 总线中的任意一台 EI 交换服务器，其中 RouteName 为服务器路由名称； 书写时，“[”和“]”需要写上，如地址： [yuxicomm.yunnanprov.tobacco.gov]2 表示上报至省级及国家局
<code>%L%[%RouteName%]</code>	EI 路由拓扑图中，从[%RouteName%]EI 服务器向叶子节点方向 %L% 级的所有 EI 服务器，不包括[%RouteName%] EI 服务器； %L% 是正整数或者 N，当为 N 时，表示从[%RouteName%]向叶子节点方向的所有 EI 服务器； 表达式[%RouteName%]表示 EI 总线中的任意一台 EI 交换服务器，其中 RouteName 为服务器路由名称； 书写时，“[”和“]”需要写上，如地址：N[tobacco.gov] 表示下发给国家局下所有节点
<code>***</code>	所有 EI 服务器节点，包括当前 EI 服务器节点

2. 服务器别名模糊指定

别名是用户自定义的唯一名字，并不限定不同 EI 服务器的别名有逻辑关系。所以指定别名时，可以用正则表达式来模糊指定多个别名。

如：`^abc\d{1,3}$`，匹配所有以 abc 开头，1 到 3 位数字结尾的别名

3. 连接器模糊指定

连接器的地址就是连接器的 ID 或者别名，都是一个单词，所以也用正则表达式来模糊指定多个地址。

4. 消息处理器模糊指定

如是指定方式是“ID”，可以使用正则表达式来指定消息处理器 ID，调用第一个 ID 与正则表达式匹配的消息处理器来处理消息。

如是指定方式是“ClassName”，必须是用确切的类名全路径来指定，不能使用正则表达式。

代码示例：

```
// 服务器指定方式
String serverAddressType = DestinationSet.ADDRESS_TYPE_ROUTENAME;
// 服务器值:路由名称,表示 Leshan 节点下所有 1 级节点
String serverAddressValue = "1[Leshan]";
// 表示是模糊指定
boolean serverAddressExact = false;

// 连接器指定方式
String clientAddressType = "ClientID";
// 连接器值: 连接器 id,正则表达式, 表示服务器下所有连接器
String clientAddressValue = "\\w*";
// 表示是模糊指定
boolean clientAddressExact = false;

//消息处理器指定方式
String processorValueType = "ID";
//正则表达式, 表示连接器下所有消息处理器 ID 以 Proecessor 结尾的处理器
String processorValue = "\\w*Proecessor";
//表示是模糊指定
boolean processorValueExact = false;

DestinationSet dest = new DestinationSet();
dest.includeAddress(
    serverAddressType,serverAddressValue,serverAddressExact,
    clientAddressType,clientAddressValue,clientAddressExact,
    processorValueType,processorValue,processorValueExact);

.....
```

3.11.2 多目的地地址集合结构说明

多目的地地址集合（DestinationSet），可以用一个 XML 结构来描述。

XML 结构：

```
<tns:DestinationSet xsi:schemaLocation=
  "destination.message.common.de.ro.icss.com
  DestinationSet.xsd"
  xmlns:tns="destination.message.common.de.ro.icss.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!--包含的目的地-->
  <tns:Include>
    <tns:Address>
      <tns:Server Type="RouteName" Exact="false"
        Value="1[Leshan]"/>
      <tns:Client Type="ClientID" Exact="false"
        Value="\w*" />
      <tns:Processor Type="ID" Exact="false"
        Value="\w*Proecessor " />
    </tns:Address>
    <tns:Address>
      .....
    </tns:Address>
    .....
  </tns:Include>

  <!--排除的目的地-->
  <tns:Exclude>
    <tns:Address>
      <tns:Server Type="" Exact="" Value="" />
      <tns:Client Type="" Exact="" Value="" />
      <tns:Processor/>
    </tns:Address>
    .....
  </tns:Exclude>
</tns:DestinationSet>
```

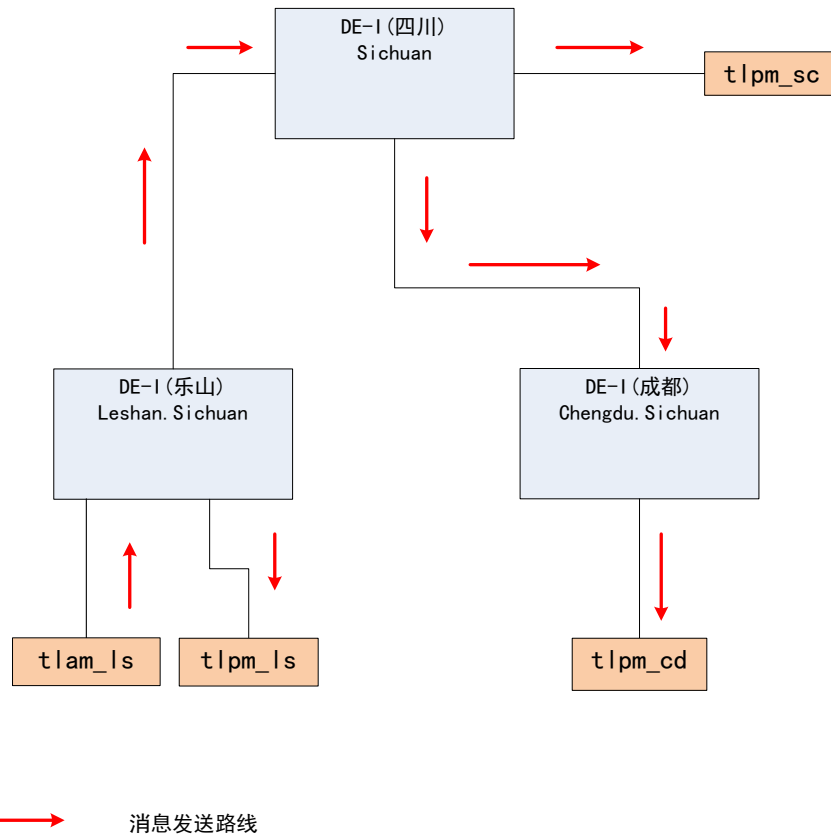
XML 元素说明：

元素名	描述	属性	描述
DestinationSet	根元素	-	-
Include	地址集合，子元素	-	-

	【Address】描述地址的详细信息		
Address	描述一个地址的信息，子元素包括【Server，Client，Processor】	-	-
Server	服务器地址描述，可以使用模糊匹配方式表示多个服务器。	Exact	TRUE:地址是明确的服务器地址 FALSE: 模糊指定
		Type	值有两种： RouteName: Value 为路由名称 Alias: Value 为服务器别名
		Value	根据 Exact、Type 的值设置对应的值，可以是服务器路由名称、别名、模糊表达式。
Client	连接器地址描述，可以使用模糊匹配方式表示多个连接器地址。	Exact	TRUE:地址是明确的 FALSE: 模糊指定
		Type	值有两种： ClientID: Value 为连接器 ID Alias: Value 为别名
		Value	根据 Exact、Type 的值设置对应的值。
Processor	连接器对应的消息处理器，可以使用模糊匹配方式表示多个。策略文件中该元素不需要填写	Exact	TRUE:地址是明确的 FALSE: 模糊指定
		Type	值有两种： ID: 连接器 ID ClassName: 消息处理器实现类全路径
		Value	根据 Exact、Type 的值设置对应的值。
Exclude	和元素 Include 相对应，表示在 Include 中的地址集合中需要排除的地址集合。格式规则和上面的【Include】一样	-	-

3.11.3 代码示例

使用 DEAgent 客户端将消息分发到多个目的地



代码示例：

```

MessageFlowHeader header = new MessageFlowHeader();
// 指定多个目的地集合
DestinationSet dest=new DestinationSet();

// 设置到 tlpmls 的地址
dest.includeAddress(DestinationSet.ADDRESS_TYPE_ROUTENAME,"Leshan.Sichuan",true,
    "ClientID", "tlpm_ls",true,
    "ID", "tlpmProcessor",true);

// 设置到 tlpmsc 的地址
dest.includeAddress("RouteName","Sichuan",true,
    "ClientID", "tlpm_sc",true,
    "ID","tlpmProcessor",true);

// 设置到 tlpacd 的地址
dest.includeAddress(DestinationSet.ADDRESS_TYPE_ROUTENAME,"Chengdu.Sichuan",true,
    "ClientID", "tlpm_cd",true,
    "ID","tlpmProcessor",true);
  
```

```
// 地址集合设置到消息头中
header.setDestinations(dest);

// 发送消息
R1DEClient rc = new R1DEClient();
rc.setMessageHeader(header);
rc.setXmlMessage("<message>...</message>");
rc.dispatchMessage();
```

3.12 客户端提供地址集合解析 API 使用说明

3.12.1 API 用途

在基于 DEAgent 的二次开发中，客户使用地址集合设置消息的目的地时，需要预先知道某地址集合对应拓扑网络中的具体服务器（或者客户端）信息，拿到这些信息后，结合具体的业务做监控、统计等。目前，支持客户端根据地址集合获取 EI 服务器信息列表、根据某路由名称获取对应的 EI 服务器信息、获取所在客户端连接的 EI 服务器信息。

通过 DEClientDestUtil 类，获取的服务器信息 DEServerNodeBean，即可取得对应的服务器信息。

需要注意的是，使用此 API 前提是客户端与 EI 服务器之间的链路时 HTTP 链路，如果使用 MQ、TLQ 链路目前还不支持。

3.12.2 代码示例

```
//new一个客户端地址集合解析工具类
DEClientDestUtil deu = new DEClientDestUtil();
//通过getDestServerList获取对应地址集合的服务器信息列表
List l = deu.getDestServerList(set);
//获取客户端所在的EI服务器的信息
DEServerNodeBean sourceServerInfo= deu.getSourceServer();
//给定EI服务器的路由名称，获取对应此路由名称的服务器信息
DEServerNodeBean serverInfo = deu.getSourceServer(routeName);
```

通过 API 获取路由信息后的使用方法参考：

```

if (l!=null){
    if (l.size()==0){
        System.out.println("根据给定的地址集合，查询服务器信息列表不存在，也就是说当前网络中没有符合此地址集合的服务器");
    }
    for (int i=0;i<l.size();i++){
        System.out.println("服务器名: " + ((DEServerNodeBean)l.get(i)).getName());
    }
} else
    System.out.println("查询服务器列表出现异常，到控制台或者客户端日志定位问题");

```

特别说明：如果获取的服务器信息（或者列表）为 NULL，则说明在调用 API 过程中出现了异常，用户需要查看一下控制台日志，如果返回的对象不为 NULL，则说明调用 API 不存在异常。

3.13 数据包导入到 RCloud 云 EI 服务服务器

RCloud 云 EI 服务提供了 SERVLET，供用户把消息数据包导入到服务器，SERVLET 名称为：ZipDataImportServlet，用户只需要通过这个 SERVLET 上传文件即可。

导入数据包时，需要注意 3 点：

- 1) 导入的数据包是一个 zip 包，并且必须是使用 RCloud 云 EI 服务提供的 SDK 导出的数据包才能导入到 RCloud 云 EI 服务服务器，否则数据包不能被 RCloud 云 EI 服务服务器正确处理。关于使用 SDK 导出数据包的代码开发说明请参考本手册中相关部分的说明。
- 2) 关于导入结果：导入成功并且 RCloud 云 EI 服务服务器处理消息成功则返回 "1"，否则返回错误信息。
- 3) 关于导入消息的消息跟踪：被导入的 RCloud 云 EI 服务服务器必须是导出数据的这个应用所对应的 RCloud 云 EI 服务服务器，否则消息跟踪将跟踪不到。

3.14 消息跟踪服务接口

RCloud 云 EI 服务提供了消息跟踪服务的接口供用户调用，用户可直接通过

程序得到消息的传输处理过程，便于监控、及时发现错误或者进行消息跟踪管控的二次开发。服务的访问地址是：

[http://\[ip\]:\[port\]/\[context\]/services/MessageTraceQueryService](http://[ip]:[port]/[context]/services/MessageTraceQueryService)

3.14.1 消息跟踪服务接口 API 列表

服务接口 API 列表如下：

服务接口 API 名称	服务接口主要功能
getSendMsgRoute	查询指定消息的发送路由路径（即消息传输过程中所经过的EI服务器节点）
getRevMsgRoute	查询指定消息的接收路由路径（即消息传输过程中所经过的EI服务器节点）
getMsgAuditList	查询某EI服务器节点上指定消息的处理过程状态
getNodeMsgList	查询某EI服务器节点上的消息列表，可根据条件进行筛选

3.14.2 消息跟踪服务接口 API 使用说明

根据不同的业务情况，可调用下列对应的服务接口 API：

- 根据业务应用提供的消息业务 ID(relationKey)访问该应用连接的交换服务器，获取该消息在交换网络中的路由路径。如果是查询发送消息的路由路径，调用 getSendMsgRoute 方法；如果是查询接收消息的路由路径，调用 getRevMsgRoute 方法。
- 根据该消息路由路径中的交换路由名，进一步获取该消息在对应经过的交换服务器上的路由和处理情况。该过程调用 getMsgAuditList 方法。
- 根据服务器路由名，根据消息标题、业务 ID(relationKey)、时间区间等筛选条件，查询该服务器上的消息列表。该过程调用 getNodeMsgList 方法。
- 以上接口的详细描述请见第三部分。

3.14.3 消息跟踪服务接口 API 详细说明

1. 查询发送消息的全局路由

根据发送消息连接器 `clientId`、消息业务 ID `relationKey` 查询消息的全局路由，将消息路由信息以 xml 字符串返回。

public String getSendMsgRoute(String clientId, String relationKey)

throws DEIException

参数名	可否为空	参数类型	备注
clientId	No	String	在开发 agent 连接器的时候，可以通过调用 Route.getCurrentRouteNode.getLocalRouteName 方法获取到 clientId(agent 需正常运行, 且 agent 与当前提供 service 的 EI 服务器相连)
relationKey	No	String	消息业务 ID (需唯一)
返回值信息描述			
满足条件的 返回值 schema	调用该 api，查询一条发送多目的地(两个目的地)的消息的路由信息，返回值示例如下：		
	<pre> <MsgRouteList> <MsgRoute> <relationKey>gov2all</relationKey> <sourceClientId>gov</sourceClientId> <sourceServerRouteName>gj</sourceServerRouteName> <targetClientId>em</targetClientId> <targetServerRouteName>sj.gj</targetServerRouteName> <serverRouteList> <serverRoutePath> <routeName>gj</routeName> <serverId>6f6900e7ecea65cda4844b7d7f80e92e</serverId> <serverName>gj</serverName> </serverRoutePath> <serverRoutePath> <serverId>6f6900e7ecea65cda4844b7d7f80e92e</serverId> <serverName>sj</serverName> <routeName>sj.gj</routeName> </serverRoutePath> </serverRouteList> </MsgRoute> <MsgRoute> <relationKey>gov2all</relationKey> <sourceClientId>gov</sourceClientId> <sourceServerRouteName>gj</sourceServerRouteName> <targetClientId>cd</targetClientId> <targetServerRouteName>gj</targetServerRouteName> <serverRouteList> </pre>		

	<pre> <serverRoutePath> <serverId>6f6900e7ecea65cda4844b7d7f80e92e</serverId> <serverName>gj</serverName> <routeName>gj</routeName> </serverRoutePath> </serverRouteList> </MsgRoute> </MsgRouteList> </pre> <p>示例说明：业务 ID 为 gov2all 的消息发送路径如下</p> <ol style="list-style-type: none"> 1、第一个 MsgRoute 节点：消息从 clientId 为 gov 的连接器发出，经过了第一个名为 gj 的 EI 服务器，然后再经过第二个为 sj 的 EI 服务器节点，最后由 clientId 为 em 的连接器接收。 2、第二个 MsgRoute 节点：消息是从 clientId 为 gov 的连接器发出，经过了第一个名为 gj 的 EI 服务器，最后由 clientId 为 cd 的连接器接收。 <p>字段说明：</p> <p>MsgRouteList：根节点</p> <p>MsgRoute：每一个 MsgRoute 中包含了一条消息的路由信息</p> <p>relationKey：消息业务 ID</p> <p>sourceClientId：发送消息的连接器 ID</p> <p>sourceServerRouteName：与发送消息连接器相连的 EI 服务器的路由名称</p> <p>targetClientId：接收消息的连接器 ID</p> <p>targetServerRouteName：与接收消息连接器相连的 EI 服务器路由名</p> <p>serverRouteList：路由列表，消息经过的 EI 服务器节点的列表，列表中的排列顺序即为消息经过的 EI 服务器的顺序</p> <p>serverRoutePath：描述消息经过的 EI 服务器节点的信息</p> <p>serverId：EI 服务器 的 uuid</p> <p>serverName：EI 服务器在管控中的名称</p> <p>routeName：EI 服务器的路由名称</p>
未找到满足条件的查询结果返回值	<MsgRouteList/>

2. 查询接收消息的全局路由

根据接收消息连接器 clientId、消息业务 ID relationKey 查询消息的全局路由，将消息路由信息以 xml 字符串返回。

public String getRevMsgRoute(String clientId, String relationKey)

throws DEIException

参数名	可否为空	参数类型	备注
clientId	No	String	在开发 agent 端应用的时候，可以通过调用

			Route.getCurrentRouteNode.getLocalRouteName 方法获取到 clientId（agent 需正常运行）
relationKey	No	String	消息业务 ID（需唯一）
返回值信息描述			
满足条件的 返回值 schema	调用该 api，查询一条接收到的消息的路由信息，返回值示例如下：		
	<pre><MsgRouteList> <MsgRoute> <relationKey>gov2all</relationKey> <sourceClientId>gov</sourceClientId> <sourceServerRouteName>gj</sourceServerRouteName> <targetClientId>cd</targetClientId> <targetServerRouteName>gj</targetServerRouteName> <serverRouteList> <serverRoutePath> <serverId>6f6900e7ecea65cda4844b7d7f80e92e</serverId> <serverName>gj</serverName> <routeName>gj</routeName> </serverRoutePath> </serverRouteList> </MsgRoute> </MsgRouteList></pre>		
	<p>示例说明：业务 ID 为 gov2all 的消息接收路径如下</p> <p>1、第一个 MsgRoute 节点：消息从 clientId 为 gov 的连接器发出，经过了第一个名为 gj 的 EI 服务器，最后由 clientId 为 cd 的连接器接收。</p> <p>字段说明：</p> <p>MsgRouteList：根节点</p> <p>MsgRoute：每一个 MsgRoute 中包含了一条消息的路由信息</p> <p>relationKey：消息业务 ID</p> <p>sourceClientId：发送消息的连接器 ID</p> <p>sourceServerRouteName：与发送消息连接器相连的 EI 服务器的路由名称</p> <p>targetClientId：接收消息的连接器 ID</p> <p>targetServerRouteName：与接收消息连接器相连的 EI 服务器路由名</p> <p>serverRouteList：路由列表，消息经过的 EI 服务器节点的列表，列表中的排列顺序即为消息经过的 EI 服务器的顺序</p> <p>serverRoutePath：描述消息经过的 EI 服务器节点的信息</p> <p>serverId：EI 服务器 的 uuid</p> <p>serverName：EI 服务器在管控中的名称</p> <p>routeName：EI 服务器的路由名称</p>		
未找到满足条件的查询	<MsgRouteList/>		

结果返回值

3. 查询指定消息在某 EI 服务器上的处理状态

根据 EI 服务器的路由名称 routeName、消息业务 ID relationKey 查询消息在指定 EI 服务器的处理情况，如接入、接出时间、是否异常等信息，将消息路由信息以 xml 字符串返回。

public String getMsgAuditList(String routeName, String relationKey)
throws DEIException

参数名	可否为空	参数类型	备注
routeName	No	String	EI 服务器路由名称
relationKey	No	String	消息业务 ID(需唯一)
返回值信息描述			
满足条件的返回值 schema	调用该 api，查询指定消息的在某 EI 服务器处理阶段的信息，返回值示例如下：		
	<pre> <getMsgAuditList> <serverId> 1fac852-136960edc2f-6f6900e7ecea65cda4844b7d7f80e92e </serverId> <serverName>gj</serverName> <routeName>gj</routeName> <auditList> <audit> <relationKey>gov2cd</relationKey> <id>9cce54-1369632ea0b-6f6900e7ecea65cda4844b7d7f80e92e</id> <date>2012-04-09 16:23:00.265 </date> <msgSize>188</msgSize> <eventType>收到接入消息</eventType> <exchangeMode>单向消息 (无响应的消息) </exchangeMode> <direction>请求消息</direction> <level>normal</level> <desc></desc> </audit> <audit> <relationKey>gov2cd</relationKey> <id>9cce54-136963300ee-6f6900e7ecea65cda4844b7d7f80e92e</id> <date>2012-04-09 16:23:06.718</date> <msgSize>188</msgSize> <eventType>发送消息</eventType> <exchangeMode>单向消息 (无响应的消息) </exchangeMode> </pre>		

	<pre> <direction>请求消息</direction> <level>normal</level> <desc></desc> </audit> </auditList> </getMsgAuditList> </pre> <p>示例说明：业务 ID 为 gov2cd 的消息在名为 gj 的 EI 服务器的处理阶段如下</p> <ol style="list-style-type: none"> 1、第一个 audit 节点:名为 gj 的 EI 服务器在 2012-04-09 16:23:06.718 接收到这条消息开始处理，日志级别 normal 表示接收过程中个处理阶段都正常。 2、第二个 audit 节点：名为 gj 的 EI 服务器在 2012-04-09 16:23:06.718 开始发送这条消息，日志级别 normal 表示发送过程中个处理阶段都正常。 <p>字段说明：</p> <p>getMsgAuditList：根节点</p> <p>serverId：EI 服务器的 uuid</p> <p>serverName：EI 服务器在管控中的名称</p> <p>routeName：EI 服务器的路由名称</p> <p>auditList：消息在 EI 服务器的处理阶段列表，以 id 作为区分</p> <p>audit：描述消息在 EI 服务器中某个处理阶段的单元节点</p> <p>relationKey：消息业务 ID</p> <p>id：即 eventId，同一个消息在 EI 服务器节点的每个处理阶段都会有 eventId 作为标示</p> <p>date：消息在 EI 服务器的处理时间</p> <p>msgSize：消息大小，单位(kb)</p> <p>eventType：事件类型。返回值中会给出中文描述，这里不做赘述。</p> <p>exchangeMode：交换类型，返回值有 “双向消息(有响应的消息)”和“单向消息(无响应的消息)”两种类型</p> <p>direction：消息方向，返回值有 “请求消息”和“响应消息”两种方向</p> <p>level：日志级别。有 normal(正常)，error(异常)，warn(警告)三种级别</p> <p>desc：备注信息。通常情况下该节点为空；当日志级别为异常时，这里是异常信息。</p>
<p>未找到满足条件的查询结果返回值</p>	<pre> <getMsgAuditList/> </pre>

4. 查询某 EI 服务器上的消息列表

根据 EI 服务器的路由名称 routeName、消息标题 messageTitle、消息业务 ID relationKey、时间、分页信息等查询条件，查询某个 EI 服务器的消息列表，将消息路由信息以 xml 字符串返回

```
public String getNodeMsgList(String routeName, String messageTitle,
    String relationKey, String startTime, String endTime, int
    currentPageNum, int rowPerPage, String orderInfo)
    throws DEIException
```

参数名	可否为空	参数类型	备注
routeName	No	String	EI 服务器路由名称
messageTitle	Yes	String	消息标题, 支持模糊查询
relationKey	Yes	String	消息业务 ID(需唯一), 支持模糊查询
start	Yes	String	根据时间段查询的开始时间点, 请注意传入的 String 格式为"yyyy-MM-dd"
end	Yes	String	根据时间段查询的结束时间点, 请注意传入的 String 格式为"yyyy-MM-dd"
currentPageNum	No	int	当前页页码
rowPerPage	No	int	每页显示消息条数, 可根据传入的条件进行灵活的分页, 但是对于一个开发人员来说输入参数 rowPerPage 是一个相对较固定的值, 所以建议在开发时, 调用该方法的时候, 建议定义一个常量 <code>static final int ROW_PER_PAGE</code> , 将该常变量的值作为 API 中 rowPerPage 的输入值。
orderInfo	Yes	String	排序信息, 目前只提供根据 date 字段的排序。如果输入值为空或非法参数, 则返回值是根据 date 字段降序排列; 若需要升序排列, 可传入"asc"

返回值信息描述

满足条件的返回值 schema	调用该 api, 查询一个 EI 服务器上的消息列表, 返回值示例如下:
	<pre><currentPageInfo> <currentPageNum>1</currentPageNum> <rowPerPage>2</rowPerPage> <totalPages>2</totalPages> <totalRows>3</totalRows> <msgList> <msgInfo> <relationKey>gov2em</relationKey> <msgTitle>gov2em</msgTitle> <date>2012-04-09 16:29:36.468</date> <exchangeMode>单向消息 (无响应的消息)</exchangeMode> <direction>请求消息</direction> </msgInfo> <msgInfo> <relationKey>gov2all</relationKey> <msgTitle>gov2all</msgTitle></pre>

	<pre> <date>2012-04-09 16:28:06.828</date> <exchangeMode>单向消息 (无响应的消息)</exchangeMode> <direction>请求消息</direction> </msgInfo> </msgList> </currentPageInfo> </pre> <p>示例说明：查询某 EI 服务器的消息列表，查询结果如下：</p> <ol style="list-style-type: none"> 1、分页信息：查询了该 EI 服务器消息列表的第 1 页，该页显示了 2 条消息记录，总页数为 2 页，总消息记录数有 3 条。 2、第一个 msgInfo 节点：该 EI 服务器在 2012-04-09 16:29:36.468 接收了一条业务 ID 为 gov2em 的消息。 3、第二个 msgInfo 节点：该 EI 服务器在 2012-04-09 16:28:06.828 接收了一条业务 ID 为 gov2all 的消息。 <p>字段说明：</p> <p>currentPageInfo：根节点</p> <p>currentPageNum：当前页页码</p> <p>rowPerPage：每页显示的消息行数</p> <p>totalPages：根据条件查询返回后的消息分页后的总页数</p> <p>totalRows：根据条件查询返回的消息据总行数</p> <p>msgList：消息概要信息列表</p> <p>msgInfo：消息概要信息，每一个 msgInfo 节点描述一条消息</p> <p>relationKey：消息业务 ID</p> <p>msgTitle：消息标题</p> <p>date：消息接收时间</p> <p>exchangeMode：交换类型，返回值有 “双向消息(有响应的消息)”和“单向消息(无响应的消息)”两种类型</p> <p>direction：消息方向，返回值有 “请求消息”和“响应消息”两种方向</p>
未找到满足条件的查询结果 返回值	<pre> </currentPageInfo> </pre>

3.14.4 代码示例

以查询指定消息的发送路由路径方法 `getSendMsgRoute(String clientId, String relationKey)`为例，通过 axis2 连接器直接调用 service 中的 `getSendMsgRoute` 获取 xml 字符串：

```
//构造参数
```



```
Object[] args = new Object[]{
    id,
    rk
};
RPCServiceClient serviceClient = new RPCServiceClient();
Options options = serviceClient.getOptions();
// 指定调用的WebService的URL
EndpointReference targetERP = new EndpointReference(
    "http://192.9.127.20:5001/rode/services/MessageTraceQueryService");
options.setTo(targetERP);
options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
options.setAction("urn: getSendMsgRoute");

serviceClient.setOptions(options);
Object[] result = serviceClient.invokeBlocking(new QName(
    "http://trace.admin.server.de.ro.icss.com/xsd", "ns1"),
    args, new Class[] { String.class });
String strXML = null;
if (result != null) {
    strXML = (String) result[0];
}
```

在获取到 xml 字符串之后，可以直接进行解析，也可以将 xml 字符串转化为 JavaBean 对象再进行使用。下面提供一种相对简便的将 xml 字符串转化为 JavaBean 的方法，该过程中会用到 xstream.jar。

1. 首先根据返回的 xml 字符串构造相应 JavaBean 对象

```
/**
 * 保存消息路由信息bean对象
 */
public class MsgRouteBean {
    private String relationKey;
    private String sourceClientId;
    private String sourceServerRouteName;
    private String targetClientId;
    private String targetServerRouteName;
    private List<ServerRoutePathBean> serverRouteList;
    /**
     * 以下省略get、set方法
     */
}
```

```
/**
 * 存储路由server节点信息的bean对象
 */
public class ServerRoutePathBean {
    private String routeName;
    private String serverId;
    private String serverName;
    /**
     * 以下省略get 、set方法
     */
}
```

2. 可以通过 XStream 的 alias(String name,Class type)方法进行 JavaBean 属性和 xml 节点字段映射，通过 from(String xml)方法来获得需要的 JavaBean 对象

```
List<MsgRouteBean> msgRouteList = new ArrayList<MsgRouteBean>();
try {
    Document document = (new SAXReader())
        .read(new ByteArrayInputStream(xml.getBytes("utf-8")));
    Element root = document.getRootElement();

    // 因为xml对象比较复杂，list中嵌套了对象，对象中又嵌套了list，所以需要一个个节点一个节点的解析
    // 把第一个list的对象节点取到，然后可以用xstream.jar的api来进行xml节点的反序列化，直接获得需要的对象
    // 注意将xml节点名与对象中属性名字不一致的地方通过alias(String name,Class type)方法进行对应
    List<Element> elemtnList = root.elements();
    Iterator<Element> msgRouteElement = elemtnList.iterator();

    while (msgRouteElement.hasNext()) {
        String msgRouteXML = msgRouteElement.next().asXML();
        MsgRouteBean bean = new MsgRouteBean();

        XStream xStream = new XStream(new DomDriver());
        xStream.alias("MsgRoute", bean.getClass());
        xStream.alias("serverRouteList", ArrayList.class);
        xStream.alias("serverRoutePath", ServerRoutePathBean.class);
```

```

        bean = (MsgRouteBean) xStream.fromXML(msgRouteXML);
        msgRouteList.add(bean);
    }

} catch (Exception e) {
    e.printStackTrace();
}

```

3.15 服务运行状态和链路状态查询

RCloud 云 EI 服务提供了服务器运行状态和链路状态查询服务接口供用户调用，用户可直接通过 WebService 的客户端获得服务器运行状态和该服务器上的链路状态，及时发现当前服务器和服务器上所配置的链路是否正常工作。查询服务的访问地址为：

[http://\[ip\]:\[port\]/\[context\]/services/EIMonitorService](http://[ip]:[port]/[context]/services/EIMonitorService)

3.15.1 接口 API

接口 API	功能描述
getNodeStatus	查询当前服务器的工作状态
getLinkStatus	查询从当前服务器起始的链路工作状态

3.15.2 使用说明

1. 服务器工作状态查询

```
public String getNoeStatus ()
```

getNoeStatus 方法以 xml 字符串形式返回服务器的当前工作状态。返回的 xml 格式如下所示：

- 服务器工作正常：

```

<?xml version="1.0" encoding="UTF-8"?>
<tns:nodeStatus xmlns:tns="http://www.chinasofti.com/ei">
    <status>OK</status>

```

```
<desc>节点状态正常</desc>
<tns:nodeStatus>
```

● 服务器工作异常:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:nodeStatus xmlns:tns="http://www.chinasofti.com/ei">
  <status>WARN</status>
  <desc>
    该节点正在运行，但启动状态存在问题，为保证正常运行请
    进一步检查该节点配置
  </desc>
</tns:nodeStatus>
```

2. 链路工作状态查询

public String getLinkStatus ()

getLinkStatus 方法以 xml 字符串形式返回当前服务器与相邻服务器之间的所有链路状态信息，返回的 xml 格式如下所示:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:linkStatus
xmlns:tns="http://www.chinasofti.com/ei">
  <link>
    <nodename>B</nodename>
    <linkname>A-B-HTTP</linkname>
    <status>OK</status>
    <time>1340694177050</time>
  </link>
  <link>
    <nodename>B</nodename>
    <linkname>A-B-FTP</linkname>
    <status>FAILED</status>
    <time>1340694177765</time>
  </link>
  <link>
    <nodename>D</nodename>
    <linkname>A-D-MQ</linkname>
    <status>OK</status>
    <time>1340694177765</time>
  </link>
</tns:linkStatus>
```

其中，每个 link 节点对应一条链路信息，如果没有配置链路信息，则 linkStatus 元素下没有子元素。nodename 为目的服务器的路由名称，linkname 为当前服务器到目的服务器之间配置的链路名称，time 为测试数据发送时间，status 为当前链路的工作状态（OK：链路工作正常;FAILED：链路工作异常;UNKNOWN：服务器刚刚启动，还没有得到链路工作状态信息，无法测试）。

第四章 开发建议

4.1 日志系统的使用

4.1.1 使用说明

RCloud 云 EI 服务提供了独立、友好的日志系统。基于 RCloud 云 EI 服务（服务器端或者 DEAgent 客户端应用）进行二次开发时，可以使用 RCloud 云 EI 服务自身的日志系统。

RCloud 云 EI 服务提供日志查看服务，可以在管理控制台根据服务器端或者应用客户端区别进行实时的日志查看。采用 RCloud 云 EI 服务日志系统，便于在远程清晰的监控程序的运行状态。同时，可以和 RCloud 云 EI 服务使用公共的日志配置文件，统一对日志级别与显示格式进行控制。如果是应用客户端开发，可以同时使用已有的日志系统，不会和 RCloud 云 EI 服务的日志产生冲突。

4.1.2 方法说明

1、trace 级别的方法：

```
public static void trace(String msg)
public static void trace(String msg, String module)
public static void trace(Throwable t)
public static void trace(Throwable t, String msg)
public static void trace(Throwable t, String msg, String module)
```

2、debug 级别的方法：

```

public static void debug(String msg)
public static void debug (String msg, String module)
public static void debug (Throwable t)
public static void debug (Throwable t, String msg)
public static void debug (Throwable t, String msg, String module)

```

3、info 级别的方法：

```

public static void info(String msg)
public static void info (String msg, String module)
public static void info (Throwable t)
public static void info (Throwable t, String msg)
public static void info (Throwable t, String msg, String module)

```

4、warn 级别的方法：

```

public static void warn (String msg)
public static void warn (String msg, String module)
public static void warn (Throwable t)
public static void warn (Throwable t, String msg)
public static void warn (Throwable t, String msg, String module)

```

5、error 级别的方法：

```

public static void error(String msg)
public static void error (String msg, String module)
public static void error (Throwable t)
public static void error (Throwable t, String msg)
public static void error (Throwable t, String msg, String module)

```

5、fatal 级别的方法：

```

public static void fatal(String msg)
public static void fatal (String msg, String module)
public static void fatal (Throwable t)
public static void fatal (Throwable t, String msg)
public static void fatal (Throwable t, String msg, String module)

```

4.1.3 代码示例

```

public class DebugUseExample {
    private final String module =
        DebugUseExample.class.getName();
    public void sampleMethod() {
        Debug.info("sample begin.", module);
        try {
            Debug.debug("debug some infomation.", module);
        } catch (Exception ex) {

```

```

        Debug.warn(ex, "there is an exception.", module);
    }
}
}

```

代码中的 `module` 变量存放当前使用类的全路径类名，使用 `module` 便于在日志中清晰的查看信息发生或错误产生的实际类。

4.2 数据源的使用

4.2.1 使用说明

RCloud 云 EI 服务为服务器端开发提供了数据源支持。

4.2.2 方法说明

1、根据数据源名称获得 Connection Provider 的实例：

```

public static DBConnectionProvider
getConnectionProvider(String datasourcePropName)
throws DBConnectionProviderException

```

2、从 Connection Provider 获得数据库连接：

```

public Connection getConnection() throws DBConnectionProviderException

```

4.2.3 代码示例

引用数据源的代码示例如下：

```

Connection conn = null;
try {
    String dataSourceName = "rone_dbcp";
    DBConnectionProvider provider =
        DBConnectionProviderFactory.
            getConnectionProvider(dataSourceName);
    conn = provider.getConnection();
} catch (DBConnectionProviderException e) {
    e.printStackTrace();
} finally {

```

```
try {  
    if (conn != null && !conn.isClosed()) {  
        conn.close();  
    }  
} catch (SQLException e) {  
    Debug.error(e,e.getMessage(),module);  
}  
}
```

代码中 `dataSourceName` 变量的值，即在管理控制台注册时的填写的数据源名称。