
第 9 章 队列和消息传递

在 4.2 中，实现了新的队列功能部件以维护多种队列类型和多种消息传递协议。队列充当用于处理具有外部源 / 目标（包括 EAI 平台和 Web 服务器）的入站和出站消息的网关。

注意要了解有关设置“消息传递框架”的信息，请继续“实现消息传递框架”一节。

为了供您参考，定义了下列术语：

- **队列** — 一个 WebSphere Product Center 构造，它充当消息的接收和传输点。脚本支持每个“队列”。
- **消息** — 由 UCCnet、EAI 平台、数据池或其它消息源提供的 XML 文档。

借助全新的队列功能部件，可以将消息作为过程的一部分，在该过程中，当导入或导出作业时，可以将状态消息发送至所有需要对象。

队列控制台

访问队列控制台

使用菜单路径“协作管理器”>“队列”>“队列控制台”，将显示“队列控制台”。

查看队列详细信息

要查看队列的详细信息，请单击队列名，将显示“队列详细信息”屏幕。提供了下列信息：

- 队列名
- 描述
- 协议
- 脚本

查看队列中的消息

“队列控制台”具有一个名为“消息”的列，该列带有指向队列已接收的消息数目的超链接。通过单击消息数目，可以查看消息的内容。

在队列中搜索消息

1. 要在队列中搜索消息，请使用菜单路径“协作管理器”>“队列”>“消息控制台”。将显示“队列消息搜索”屏幕。
2. 为下列字段选择值

- 起始到达日期
- 结束到达日期

3. 单击“搜索”，结果将显示在下面的“队列消息”表中。

创建队列

1. 使用菜单路径“协作管理器”>“队列”>“新建队列”。

2. 输入必需的信息：

队列名：输入队列的名称

描述：输入队列的描述

协议：从将用来使外部源与队列相关的消息传递协议的列表中进行选择。

脚本：从预先编写的脚本的列表中进行选择（可以运行这些脚本以将消息传递至该队列或者从该队列传递消息）。目标 / 源的典型消息包括：

- 受工作流步骤中的输入或输入脚本支持的工作流协作区
- 受预处理、后处理或后保存脚本支持的目录

实现消息传递框架

WebSphere Product Center 的消息传递框架的实现允许与以下目标 EAI 平台集成：

- IBM WBI
- SeeBeyond
- Tibco
- WebMethods

选择的平台应该为程序提供可靠的传输机制和一致的接口，以使那些程序能够跨不同的系统或平台相互通信。

WebSphere Product Center 的消息传递框架已设计成支持下列过程：

- 接收包含项集合的消息，包括在接收之后提供确认消息
- 发送包含项集合的消息，包括在传输之后接收确认消息

WebSphere Product Center 提供了对 XML 文档进行语法分析以及创建 XML 文档、将消息发送至 EAI 平台队列以及从 EAI 平台队列收集消息的能力。消息被定义为由外部源提供的 XML 文档。此功能完全可供 WebSphere Product Center 脚本编制引擎访问。为了能够在 WebSphere Product Center 与 EAI 平台适配器之间进行交互，有必要安装利用此功能的脚本。

WebSphere Product Center 支持队列，后者是一个充当消息传递接收点和传输点的构造。WebSphere Product Center 队列充当用于处理具有外部源 / 目标的进站和出站消息传递的网关，并且每个队列都受 WebSphere Product Center 脚本编制操作支持。

队列提供了下列功能：

允许设置消息传输协议以使外部消息源与队列相关，并提供了下列消息传输协议：

- MQ
- JMS 点到点
- JMS 发布和预订
- HTTP
- HTTP/S

运行脚本以将消息发送至队列或者从队列发送消息。典型的消息目标 / 源包括：

- 工作流协作区 — 受工作流步骤中的输入或输出脚本支持
- 目录 — 受预处理、后处理或后保存脚本支持

下面列表描述了消息传递功能的框架：

- 外部源将消息发送至 EAI 平台。根据外部源的不同，处理此操作的方式也会有所变化，并且此方式不会对 WebSphere Product Center 检索消息的方式产生影响。
- EAI 平台的设置应该包括进站和出站队列。此设置是由第三方完成的，为了检索和发送消息，WebSphere Product Center 所需了解的信息只有 WebSphere Product Center 将要访问的队列的标识。
- 在将文件放到 EAI 平台队列中时，在 WebSphere Product Center 上设置的相应队列能够通过受支持的协议（MQ、HTTP/s 或 JMS）来检索消息。
- 通过使用 WebSphere Product Center 脚本，将对消息体进行语法分析以获取消息类型、消息标识和消息源。将把此信息传递至为了包含所有消息而创建的目标 WebSphere Product Center 目录。
- 将使用一个工作流来跟踪在 D 中发生的事件，在成功地记录消息之后，一个事件触发“确认工作流”以将新记录从消息目录中检出到协作区设置中。
- “确认工作流”将确认消息发送至消息源以进行确认。完成此操作后，“确认工作流”将该消息重新检入到目录中。

为了使此过程能够发挥作用，必须构建消息传递框架。

构建消息传递框架

以下过程概述了集成 EAI 平台框架的建议方法。可以根据特定的需求来定制此过程。

接收消息

本节描述接收包含项集合的消息的过程，包括在接收该消息之后提供确认消息。以下两个过程（“设置”和“运行时”）支持接收消息的过程。对于大多数目的，此过程具有一般性和普遍的适用性。

设置

1. 业务流程技术分析员创建下列内容：
2. 业务流程技术分析员构建包含脚本的进站队列。脚本支持三项功能— 接收消息、对消息体进行语法分析以及传递消息。

接收消息

脚本的消息接收部分支持以下功能：

- 通过受支持的协议（包括 MQ、HTTP/S 或 JMS）从源获取消息
- 对消息体进行语法分析以获取消息类型、消息标识及消息源
- 在消息目录中创建包含该消息类型、消息标识、发送方标识和日期时间的记录
- 对“确认工作流”触发事件。要了解“确认工作流”的功能，请参阅以下内容。

消息体语法分析

脚本的消息体语法分析部分支持以下功能：

- 以参数形式包括源到目标映射名和目标目录名
- 针对每个源到目标映射名和目标目录名，对消息体进行语法分析以填充项集合

消息传递

脚本的传递部分支持以下功能：

根据目标目录，对项集合添加 / 修改 / 删除项。

- 业务流程技术分析员设置用于发送接收确认消息的“确认工作流”。此工作流具有以下功能：
- 将新记录从消息目录检出到“确认工作流”中的一个步骤中的协作区中。
- “确认工作流”中的下一个步骤发送一条确认消息给消息源，该消息包含消息标识、发送方标识、日期时间以及消息源所需的任何必需命令（例如 Received）。
- “确认工作流”中的下一个步骤检入消息目录记录。

运行时

在正确地配置设置之后，应该会发生下列运行时事件。

1. 队列通过队列脚本的消息接收部分接收消息

2. 队列脚本的消息接收部分对消息体进行语法分析以获取消息类型、消息标识和发送方标识
3. 队列脚本的消息接收部分在消息目录中创建记录，该记录包含消息类型、消息标识、发送方标识和日期时间
4. 队列脚本的消息接收部分将新记录从消息目录检出到“确认 workflow”中的一个步骤中的协作区中。
5. “确认 workflow”中的下一个步骤发送一条确认消息给消息源，该消息包含消息标识、发送方标识、日期时间以及消息源所需的任何必需命令（例如 `Received`）。
6. “确认 workflow”中的下一个步骤检入消息目录记录。
7. 消息脚本的消息体语法分析部分针对每个映射名和目标目录名来对消息体进行语法分析（使用下面叙述的新脚本操作来进行语法分析）以填充项集合。
8. 队列脚本的传递部分使用用于添加 / 修改 / 删除目录的现有脚本操作来根据目标目录对项集合添加 / 修改 / 删除项。

创建 WebSphere Product Center 入站 / 出站队列

WebSphere Product Center 入站和出站队列是使用“队列控制台”创建的。在创建任何队列之前，必须从“脚本控制台”中创建触发器脚本。触发器脚本将显示在“新建队列”屏幕的“触发器脚本路径”下拉字段中。

1. 在“队列控制台”中，单击**新建**。
2. 在“队列详细信息”屏幕中，输入队列名和描述；选择协议和触发器脚本路径。触发器脚本是在类型为“消息队列处理器”的“脚本控制台”中创建的。
3. 单击**保存**。

消息传递脚本操作

WebSphere Product Center 脚本操作提供了一种灵活的方法来编写 WebSphere Product Center 脚本应用程序，并且每个脚本操作的自变量有定义能力。本节中标识的下列脚本操作用来支持 WebSphere Product Center 通过使用 MQ 或 JMS 来支持的消息传递功能。这些方法允许从外部队列导入消息以及将消息导出到外部队列中。

注意：本节中列示的脚本编制操作会更改。要了解最新的脚本操作，请参阅“脚本沙箱”。

MQ 脚本编制操作

当创建脚本应用程序时，脚本操作 `mqGetQueueMgr` 将返回一个句柄给 `MQQueueManager`，借助该句柄，可以在调用 `mqDisconnect` 以释放该句柄之前执行多个 MQ 操作。

`mqGetQueueMgr`

- 原型： `MQQueueManager mqGetQueueMgr(String hostname, String port, String channel, String queueMgrName)`
- 描述：使用给定的属性来创建并返回新的 MQ 队列管理器。

`mqDisconnect`

- `void MQQueueManager::mqDisconnect()`
- 与给定的队列管理器断开连接。

`mqSendTextMsg`

- 原型： `MQMessage MQQueueManager::mqSendTextMsg(String msgText, String queueName, String queueOpenOptions, String messagePutOptions)`
- 描述：通过 `queueName` 发送在 `String msgText` 中提供的消息。返回 `MQMessage`

注意：如果尝试使用 `mqSendReply` 来发送对给定消息的回复，并且使用了 `mqSendTextMsg`，就会返回错误。要避免这种情况，请使用 `mqSendTextMsgWithReply`

`mqSendTextMsgWithReply`

- 原型： `MQMessage MQQueueManager::mqSendTextMsgWithReply (String msgText, String queueName, String replyQueueName, String queueOpenOptions, String messagePutOptions)`
- 描述：通过 `queueName` 发送在 `String msgText` 中提供的消息。指定了回复队列。返回 `MQMessage` 对象。

`mqGetTextFromMsg`

- 原型： `String mqGetTextFromMsg(MQMessage mqMessage)`
- 描述：返回一个字符串，该字符串包含 `MQMessage` 的整个内容，包括头。

`mqGetReceivedMsg`

- 原型： `MQMessage MQQueueManager::mqGetReceivedMsg(String queueName, String queueOpenOptions, String messageGetOptions)`
- 描述：从 `queueName` 接收消息。返回消息（作为 `MQMessage`）或空。

注意：检索消息时，将把它们从队列中除去。除非指定了消息标识，否则将获取队列中的第一条消息。

`mqSendReply`

- 原型: MQMessage MQQueueManager::mqSendReply(MQMessage receivedMsg, String msgText, String passedInQueueOpenOptions, String passedInMessagePutOptions)
- 描述: 发送对给定消息的回复, 并且不指示成功或失败。

mqSendReplyWithStatus

- 原型: MQMessage MQQueueManager::mqSendReplyWithStatus(MQMessage receivedMsg, String msgText, String status, String passedInQueueOpenOptions, String passedInMessagePutOptions)
- 描述: 发送对给定消息的回复, 并且设置反馈字段以指示给定的状态。状态必须为下列其中一项 (大写或小写): SUCCESS、FAIL、VALCHANGE、VALDUPES、MULTIPLE_HITS、FAIL_RETRIEVE_BY_CONTENT、BO_DOES_NOT_EXIST、UNABLE_TO_LOGIN、APP_RESPONSE_TIMEOUT 和 NONE。

注意: 只能使用一个状态值。

mqGetXMLMessageContent

- 原型: String mqGetXMLMessageContent(String orgXmlMsg)
- 描述: 废弃位于输入字符串开头的任何垃圾以获取 XML 文档。更准确地说, 行为如下所示: 如果输入字符串具有 A+B 格式, 其中 B 是有效的 XML 文档, A 是任何 (可能是空的) 字符串, 则此操作返回 B。否则, 返回 NULL。

注意: 使用此方法来对进入消息进行解析。

mqGetResponseToMsg

- 原型: MQMessage MQQueueManager::mqGetResponseToMsg(MQMessage outgoingMessage, String queueOptions, String messageOptions)
- 描述: 从给定队列获取对给定消息的响应。

mqGetMessageDiagnostics

- 原型: String mqGetMessageDiagnostics(MQMessage message)
- 描述: 返回一个字符串, 该字符串包含关于给定消息的诊断信息。

mqGetMessageId

- 原型: String MQMessage::mqGetMessageId()
- 描述: 将给定消息的标识作为包含十六进制数字的字符串返回。

mqGetReceivedMsgByMessageID

- 原型: MQMessage MQQueueManager::mqGetReceivedMsgByMessageID(String queueName, String messageId, String passedInQueueOpenOptions,

String passedInMessageGetOptions)

- 描述：在给定队列中查找具有给定消息标识的消息。该标识是在包含十六进制数字的字符串 中传递的。如果在给定的队列中没有这样的消息，则返回 NULL。

JMS 脚本操作

当创建脚本应用程序时，脚本操作 `jmsGetConnectionFactory` 将返回一个句柄给 `QueueConnectionFactory`，借助该句柄，可以在调用 `jmsDisconnect` 以释放该句柄之前执行多个 JMS 操作。

`jmsGetContext`

- 原型：Context jmsGetContext(String url, String jndiFactory)
- 描述：创建 JMS 上下文。

`jmsGetConnectionFactory`

- 原型：QueueConnectionFactory Context::jmsGetConnectionFactory(String jmsFactory)
- 描述：创建并返回具有指定上下文的 jms 连接工厂。

`jmsGetMQConnectionFactory`

- 原型：QueueConnectionFactory jmsGetMQConnectionFactory(String mqQueueManager, String mqHostname, String mqChannel, Integer mqPort)
- 描述：创建并返回用于与 MQ 队列进行通信的 JMS 连接工厂。注意，不需要 Context 就可以获取 MQ 连接工厂，但是，需要 Context 才能连接至其它 JMS 队列。

`jmsGetQueueByName`

- 原型：javax.jms.Queue jmsGetQueueByName(Context ctx, String name)
- 描述：从给定的 JNDI 名称和 Context 返回 javax.jms.Queue 对象。

`jmsGetQueueConnection`

- 原型：QueueConnection QueueConnectionFactory::jmsGetQueueConnection()
- 描述：从给定的连接工厂返回 JMS 队列连接。

`jmsGetQueueSession`

- 原型：QueueSession QueueConnection::jmsGetQueueSession()
- 描述：从给定的连接工厂返回 JMS 队列连接。

`jmsDisconnect`

- 原型：void QueueSession::jmsDisconnect(QueueConnection qcon)

- 描述：与给定的队列管理器断开连接。

jmsCreateTextMsg

- 原型：Message QueueSession::jmsCreateTextMsg(String msgText)
- 描述：使用带有所提供的文本的 QueueSession 信息来创建新的 JMS TextMessage。

jmsSendMsg

- 原型：Message QueueSession::jmsSendMsg(Message msg, String queueName[, HashMap properties, Message messageToReplyTo])
- 描述：通过具有名称 queueName 的队列发送消息 MSG 并返回该 MSG，或者返回 NULL。如果提供了 MESSAGESTOREPLYTO，则从中读取对队列和消息标识的回复。PROPERTIES 是从字符串键到字符串值的映射。有三个特殊键：“WPC_REPLY_TO_QUEUE”、“WPC_COPY_CORRELATION_ID_BYTES”和“WPC_COPY_CORRELATION_ID”。如果提供了 WPC_REPLY_TO_QUEUE，则它覆盖 QUEUENAME 或者所提供的 MESSAGESTOREPLYTO 中的回复队列。MESSAGESTOREPLYTO 中的回复队列覆盖 QUEUENAME。“WPC_COPY_CORRELATION_ID”和“WPC_COPY_CORRELATION_ID_BYTES”将 MESSAGESTOREPLYTO 中的相关标识复制到 MSG。这两者都可以被提供。它们的值需为布尔值（与上面描述的字符串相反）。

jmsReceiveMsgFromQueue

- 原型：JMSMessage QueueSession::jmsReceiveMsgFromQueue(javax.jms.Queue queue, Integer timeout[, String messageSelector, JMSMessage messageToReceiveReplyFor])
- 描述：接收 JMS 消息。在 TIMEOUT 毫秒之后超时。如果 INBOUNDQUEUE 不为 NULL，则在该队列中进行查找。如果 INBOUNDQUEUE 为 NULL，并且 MESSAGESTORECEIVEREPLYFOR 不为 NULL，则在 MESSAGESTORECEIVEREPLYFOR 的 Reply-To 字段中定义的队列中进行查找。如果 INBOUNDQUEUE 为 NULL，并且 MESSAGESTORECEIVEREPLYFOR 也为 NULL，则抛出 AustinException。我们现在知道哪个队列将被使用。如果 MESSAGESELECTOR 和 MESSAGESTORECEIVEREPLYFOR 都为 NULL，则从该队列中选择第一条消息。否则，从满足由 MESSAGESELECTOR 和 MESSAGESTORECEIVEREPLYFOR 定义的所有条件的队列（如果有的话）中选择第一条消息。如果 MESSAGESTORECEIVEREPLYFOR 不为 NULL，则拒绝相关标识不等于 MESSAGESTORECEIVEREPLYFOR 的消息标识的任何消息。如果 MESSAGESELECTOR 不为 NULL，则拒绝任何不满足 messageSelector 中定义的条件消息。如果找不到适合的消息，则返回 NULL。

jmsGetTextFromMsg

- 原型: `String Message::jmsGetTextFromMsg()`
- 描述: 返回一个字符串, 该字符串包含 JMS 的整个内容, 包括头。

`jmsGetMessageID`

- 原型: `String Message::getJMSMessageID()`
- 描述: 返回包含 JMS 消息标识的字符串。

`jmsGetMessageCorrelationID`

- 原型: `String Message::getJMSMessageCorrelationID()`
- 描述: 返回包含 JMS 消息的相关标识的字符串。

`jmsGetMessageProperties`

- 原型: `HashMap Message::getJMSMessageProperties()`
- 描述: 返回从字符串属性名到那些优先级的字符串值的散列映射。

`jmsSendMsgToQueue`

- 原型: `JMSMessage QueueSession::jmsSendMsgToQueue (JMSMessage msg, javax.jms.Queue outboundQueue [, HashMap properties, JMSMessage messageToReplyTo,])`
- 描述: 发送消息 MSG 并返回 MSG, 或者返回 NULL。除非 OUTBOUNDQUEUE 为 NULL, 否则将消息发送至由 OUTBOUNDQUEUE 指定的队列。如果 OUTBOUNDQUEUE 为 NULL, 则将 MSG 发送至 MESSAGESTOREPLYTO 的回复队列 (如果提供了 MESSAGESTOREPLYTO 的话)。如果 OUTBOUNDQUEUE 为 NULL 并且未提供 MESSAGESTOREPLYTO, 则抛出 `AustinException`。如果提供了 MESSAGESTOREPLYTO, 则从中读取消息标识。PROPERTIES 是从字符串键到字符串值的映射。有一个特殊的 (非 JMS) 键: `WPC_INCOMING_REPLY_QUEUE`。`WPC_INCOMING_REPLY_QUEUE` 指示一个 `javax.jms.Queue` 对象, 外部应用程序应该将对此消息的回复发送至该对象。

`jmsSetMessageText`

- 原型: `void Message::setJMSMessageText(String msgText)`
- 描述: 为 JMS TextMessage 设置所提供的文本。仅支持 JMS TextMessage 类型。

Web Service

创建新的 Web Service

使用菜单路径: **协作管理器 > Web Service > 新建 Web Service**。将显示 “Web Service 详细信息” 屏幕。

将适当的信息输入到下列字段中：

Web Service 名称	输入 Web Service 的名称。此名称将成为 SOAP 服务的 URL 的一部分。它一定不能包含任何空格。例如，
Web Service 描述	输入 Web Service 的描述。
协议	用于 Web Service 的协议。当前，基于 HTTP 的 SOAP 是唯一受支持的协议。缺省值为“SOAP_HTTP”。
URL	提供可以访问该服务的 URL。在保存了该 Web Service 之后，会自动填充此字段。
WSDL URL	可以用来访问 Web Service 的 WSDL 的 URL。在保存了该 Web Service 之后，会自动填充此字段。
WSDL	输入此服务的 WSDL。WSDL 文档以 XML 格式描述了该服务的接口、URL 和协议。虽然必须手工输入此文档，但是我们下面提供了样本 WSDL 文档。必须输入有效 XML 以便成功保存该 Web Service。
实现脚本	输入实现此服务的 Trigo 脚本。在数组变量“soapParams”中提供了该服务的输入参数。该服务的返回值必须为字符串，并且可以写至“out”写程序变量。要返回 SOAP 故障，将故障代码写至“soapFaultCode”写程序变量，并将故障消息写至“soapFaultMsg”写程序变量。在下面提供了样本实现脚本。
存储请求？	如果选择此项，则 Trigo 将把所有输入请求的参数存储在文档库中。将可以从“事务控制台”中获取它们。
存储应答？	如果选择此项，Trigo 将把所有响应中的内容存储在文档库中。将可以从“事务控制台”中获取它们。
部署的	如果选择此项，则将部署服务。否则，此服务将不可用。

样本实现脚本和 WSDL 文档

样本实现脚本和 WSDL 文档的以下设计会检查输入 SOAP 请求的参数的数目。如果刚好有 4 个参数，则会返回列表示这些参数的字符串。如果多于或少于 4 个参数，则会抛出 SOAP 故障。

实现脚本

```
nParams = soapParams.size();
if (nParams != 4)
{
    soapFaultCode.writeln("WRONG_NUM_PARAMS");
    soapFaultMsg.writeln("Wrong number of parameters.
    This service requires 4.  You have " + nParams + " parameters.");
}
else
{
    out.writeln("Success.");
    for (i = 0; i < nParams; i++)
    {
        out.writeln("Parameter " + (i + 1) + " is " + soapParams[i]);
    }
}
```

```

    }
  }
}
WSDL
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=
"http://my.trigo-instance.com/soap/services/CheckParams"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl=
"http://my.trigo-instance.com/soap/services/CheckParams"
    xmlns:intf=
"http://my.trigo-instance.com/soap/services/CheckParams"
    xmlns:soapenc=
"http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:message name="invokeRequest">
    <wsdl:part name="param1" type="xsd:string"/>
    <wsdl:part name="param2" type="xsd:string"/>
    <wsdl:part name="param3" type="xsd:string"/>
    <wsdl:part name="param4" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="invokeResponse">
    <wsdl:part name="invokeReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="CheckParams">
    <wsdl:operation name="invoke" parameterOrder="param1 param2 param3
param4">
      <wsdl:input message="intf:invokeRequest" name="invokeRequest"/>
      <wsdl:output message="intf:invokeResponse" name="invokeResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CheckParamsSoapBinding" type="intf:CheckParams">
    <wsdlsoap:binding style="rpc" transport=
"http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="invoke">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="invokeRequest">
        <wsdlsoap:body encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"
            use="encoded"/>
      </wsdl:input>
      <wsdl:output name="invokeResponse">
        <wsdlsoap:body encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
            namespace=
"http://my.trigo-instance.com/soap/services/CheckParams"
            use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CheckParamsService">
    <wsdl:port binding="intf:CheckParamsSoapBinding"
        name="CheckParams">
      <wsdlsoap:address location=
"http://my.trigo-instance.com/soap/services/CheckParams"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```