用友 yonyou

用友UAP 统一应用平台 Unified Application Platform

UAP6.3 信息交换平台 关键技术说明

用友软件股份有限公司

2013 年 7 月 11 日

文档控制

更改记录

日期	作者	版本	更改参考/备注
2011-12-01		1.0	

审校

日期	审校人	版本	审校意见

批准

姓名	职位	签字

分发人员

序号	姓名	分发地点

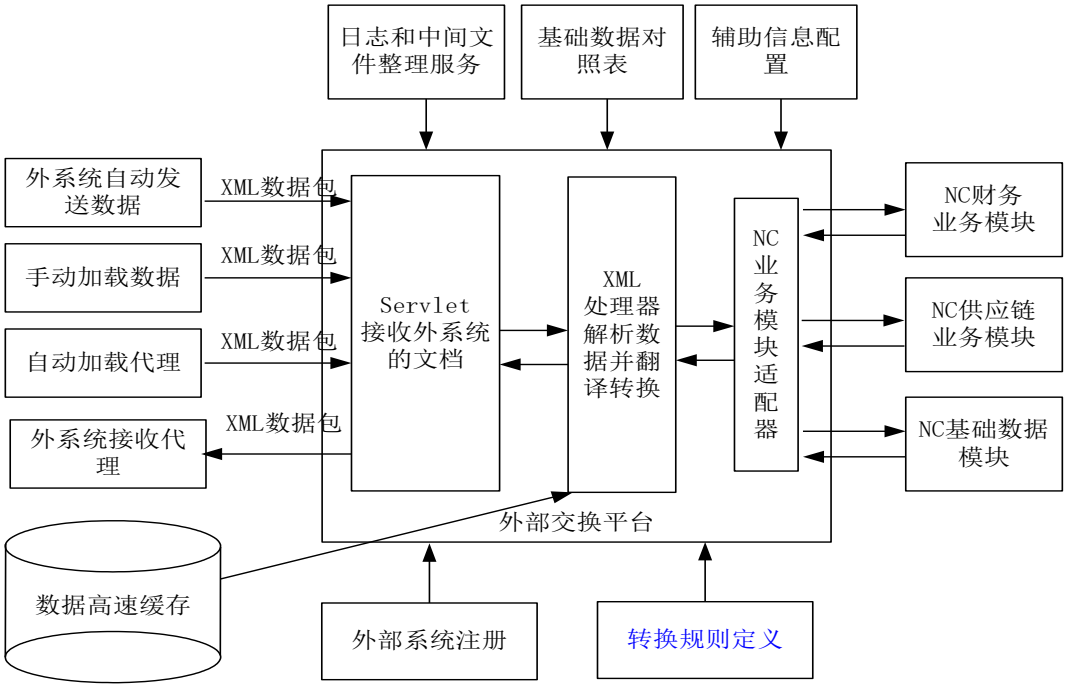
版权

目录

1 前言

信息交换平台主要用于外部系统与 NC 系统进行数据交换。利用信息交换平台，可以将外系统的基本档案和业务数据发送到 NC 系统中，并进行相关的业务操作（如审批、弃审等），也可以通过信息交换平台查询 NC 系统中的基本档案、业务数据。

1.1. 信息交换平台总体结构



利用信息交换平台，通过传输 XML 可以将外系统的基本档案和业务数据发送到 NC 系统中，同时通过发送 XML 格式的查询条件导出 NC 系统的数据（需业务插件支持），导出的数据可以附着在回执文件中，也可以直接向外部系统回发 HTTP 请求。

1.2. 信息交换平台功能特点

- 采用 XML 格式作为统一的数据交换标准，为数据访问提供简便、 统一的模式。XML 格式在数据表达和描述方面有着很大的优势，逐渐成为业界的标准，采用 XML 格式作为交换标准格式可以很好的保护企业投资。

- 面向服务的架构。这使得第三方系统可以随时随地向 NC 系统发送相关的业务数据，NC 内部的预警服务及工作流引擎使得 NC 系统可以在合适的时候向第三方系统传送需要的数据，并且满足第三方系统的格式规范。

- 灵活配置。数据转换的规则可灵活定义，独立于应用集成和业务逻辑，也就是说根据不同的外部数据结构，直接通过修改交换规则文件的定义，即可达到各种异构数据无缝集成的目的。

- 自由扩充。对于标准产品不支持的业务单据，如果有集成需求，信息交换平台提供了向导式的二次开发工具，集成了所有与二次开发相关的功能及配置，支持动态部署，可以在用户环境上进行快速有效的开发。

1.3. 信息交换平台 V50 版新增功能

与信息交换平台 V3 序列产品相比，V50 版信息交换平台在功能上得到了进一步的完善，在易用性上也有了很大改进。

1) 单据交换规则的定义更加丰富和灵活。可以为某一单据模型中的某一字段定义路径，也可以为一个集合中的实体元素定义路径，甚至可以为某个字段定义 XML 结构查询的公式。这些都得益于对 XPath 功能的模拟，使得 XML 文件间结构转换的能力更强，但这个交换规则的学习成本可能比较大，后续版本需要图形化和简洁化。

2) 为基于信息交换平台的自定义单据的集成提供了一个快速开发工具。这个工具以向导方式将集成一个自定义单据所要做的工作贯穿起来，包括：单据信息注册、校验文件生成、样本数据导出、辅助信息格式配置、业务插件类代码生成和编写，甚至还包括业务插件类代码的实时编译和部署，样本数据的导入测试和结果展示等。

3) 启用 NCV50 新缓存机制。这不但使得外部系统设置、辅助信息设置和基础数据对照表的设置能够实时地作用于外部数据导入过程中，而且对于基本档案数据的访问也实现了实时性，效率得到了更高的优化。

4) 基础数据对照导入功能更加丰富。基础数据对照在 V35 版根据基本档案自动增加

的基础上，新增了基础数据对照的 XML 文件增加和 EXCEL 文件增加的功能。

5) 信息交换平台总体参数配置功能更加丰富和完整。这部分配置主要包括单据导入方式、接收公司匹配规则、导入过程是否记录中间文件、回执和导出文件编码格式、设置客户端 IP 范围等，这些功能的可配置性极大地提高了信息交换平台的灵活性和可扩展性。

6) 对单据导入过程中的并发程度进行控制。对于某一单据类型，一共给出了四种并发控制程度，实施人员可根据具体情况设置合理的并发控制级别，避免单据并发导入引发的错误。

7) 外系统数据实现后台异步发送。借助预警平台的定时触发功能，系统可以自动将保存在服务端某个目录下的外系统数据导入 NC 系统中，实现了异步驻留的数据发送方式。

8) 对数据导入过程中的异常进行了重新划分，对于错误信息进行了编码。前者带来的好处就是错误提示信息更加明确化，后者主要用于第三方系统程序内部识别导入过程中是否出现错误以及具体的错误信息。

9) 进一步约束信息交换平台导入单据的语义。对每个单据需要定义一个组织字段，用于确保往某个接收方，比如说 A 公司做加载数据时，导入的数据能真正进入 A 公司，这个组织字段一般是公司或者主体账簿。为单据定义组织字段之后，如果数据里相应组织字段的值不为空，则要求其值与接受方保持一致，如果数据里相应组织字段的值为空，则取接受方的内容为组织字段的值。

10) 易用性改进。【手动加载界面】对文件目录、回执目录、目标 URL 地址、加载成功转移目录的配置增加记忆功能，就是说以相同公司相同用户登陆时，上述各项的配置内容默认显示为最后一次用户设置的值，避免用户每次都需要重新设置。【交换规则定义】增加了查找、定位字段的功能，单据交换规则树上的右击菜单也更加简练明确，同时为字段的导入导出公式定义增加新版本的公式编辑器，定义公式更加方便简洁。

上述功能我们在接下来的内容中会具体提到。

1.4. 信息交换平台 V55 版新增功能

- 最大传输上限界面最大 20M
- 业务插件扩展功能 见第七章 扩展
- 基础数据对照界面 EXCEL 内外对照数据导入分为两列，其中第一列为外系统值，第

二列为 NC 所对应值，系统会自动检查 NC 值的合法性，不合法则置空。

- 交换规则增加合并功能
- Ufinterfacevo 填充 NC 操作员信息
- 增加日志统计和输入流监控，便于进行日志分析，定位网络错误

1.5. 信息交换平台 V6x 版新增功能

- 信息交换平台全面支持 NCV6x 元数据进行数据处理。
- 增加自定义翻译器注册界面，可以建立自定义的翻译器。
- 插件开发向导支持通过元数据和 VO 类生产交换规则。
- 对枚举类型数据的自动转换。

2. 实施简介及相关注意点

NC 信息交换平台的主要功能就是将不同外系统的数据导入 NC 系统。由于不同系统之间对于相同单据的数据在表示上千差万别，需要信息交换平台对单据的 XML 格式的外系统数据进行格式转换和数值翻译，然后将转换后生成的 XML 格式的 NC 标准数据实例化为 NC 系统对应单据的数据对象，并调用业务模块的接口（服务）进行保存或者更新。

NC 系统在后台有一个 Servlet 伺候服务，等待并处理客户端的 POST 方法请求（信息交换平台提供界面客户端和预警客户端，对于同步集成方式，外系统可以自定义发送客户端）。客户端往根据 Servlet 的 URL 建立的连接的输出流中写待发送数据文件的内容，而后台服务端则从请求的输入流中读取数据文件内容，然后再进行后续处理。关于客户端发送数据的具体内容请参考 2.3 节。

2.1 节介绍了要将某个外系统的某种单据类型的数据文件导入至 NC 系统的最简单方法。其余小节介绍了在实施数据导入过程中容易混淆而需要注意的地方，或者是一些常用的工具等。

2.1. 实施方法简介

2.1.1. 外系统数据导入的一般步骤

外系统根据单据类型将相同单据类型的数据组织在一个 XML 文件中，然后将其发送到 NC 系统的某个账套下的某个组织中。一般来说，如果需要发送某种单据类型的数据至 NC 系统中，需要如下几步：

一、注册外部系统。如果不存在可用的外系统的话，请在“外部系统信息设置”界面中注册一个外部系统。

二、准备外系统数据。这份数据可能是外系统直接输出的，也可能是二次开发人员通过写程序从第三方系统数据库中抓出来的，或者由 Excel 格式或其他格式文件转换过来的。写这份数据时，可以参考 NC 安装盘附带的 XML 模板，XML 文件头或者说文档头的属性项请参照 2.2 节详细说明。

三、配置辅助信息（可选）。如果要导入的单据数据需要辅助信息配置，在“辅助信息

配置”界面根据外部系统、单据类型、接收组织为此次文件发送配置辅助信息。

四、设置基础数据对照（可选）。如果要导入的单据数据需要作基础数据对照（对于需要参照基本档案的字段，如果其值不能按名称或者编码自动翻译过来的话，在导入过程中系统会自动提示必须为该值做基础数据对照），在“基础数据对照”界面根据需要参照的外部系统、需要参照的基本档案、组织为需要对照的值做基础数据对照。

六、配置 Servlet 的 URL 地址。将要发送至的帐套编码作为 account 属性值写入要发送到的 Servlet 的 URL 中（或者写入 XML 文件的头中）；将接收集团编码作为 groupcode 属性写入要发送到的 Servlet 的 URL 中（或者写入 XML 文件的头中）；将接收组织的代码作为 orgcode 属性写入要发送到的 Servlet 的 URL 中（或者 XML 文件的头中）。详细说明请参见 2.2 节。

最后，就可以利用客户端触发该单据类型的数据文件的发送了。

NC 信息交换平台内置了常用单据类型数据的导入功能，对于这些单据类型，如果外系统数据遵照了这些单据类型的交换规则定义(即如果外系统的数据严格按照 NC 安装盘附带的相应单据的 XML 模板生成)，那么按上述步骤相对比较简单地就可以导入数据。但是如果外系统数据未能遵照单据类型的交换规则定义，则需要修改该单据类型的交换规则定义。做法就是将该单据的普通交换规则定义文件另存为特定该外部系统的交换规则定义文件，并根据单据交换规则定义规范修改该特定交换规则定义文件，具体需要参见第三章[单据交换规则定义]，然后再行导入。对于非系统内置或者自定义的单据的导入，需要基于信息交换平台做二次开发，这部分内容可参见第四章。

2.1.2. 信息交换平台服务器端文件目录结构

信息交换平台服务端的目录结构安排如图 2.1.1 所示，其中 NC_COMMON 为安装盘根目录，也是中间件的工作目录。

从上图可以看出，信息交换平台配置文件目录 pfxx 与 webapps 一样同在安装盘根路径下，其中：

pfxxtemp 目录，存贮信息交换平台接受到的原始数据文件、转换翻译完毕的标准 XML 文件、传送失败的文件。

exportbills 目录，存放信息交换平台发送给外系统的数据文件。

billdefine 目录，存放所有需要交换的档案和单据的交换规则文件。

auxiregister 目录，每个模块在这个目录下注册一个文件，文件的内容是模块所涉及单据的辅助信息格式。详细情况参考 4.3 节辅助信息项设置。

businessprocessor 目录，每个模块在这个目录下注册一个文件，文件的内容是模块所涉及单据在信息交换平台的注册信息，如单据类型、业务插件类名称、元数据 ID、单据加锁级别等。详细内容参考 4.1 节注册单据相关信息。

globalset.xml 用于存放信息交换平台的全局参数，如默认帐套、单篇最大传输上限等。

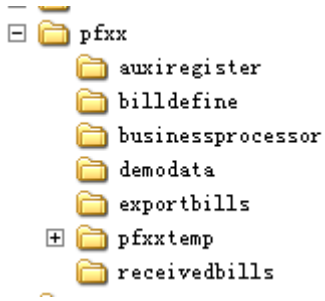


图 2.1.1 信息交换平台服务端文件目录结构

2.2. Servlet 的 URL 地址参数与 XML 交换文档头属性的关系

在外系统的单据数据的导入过程中需要用到一些初始化信息，如帐套、接收集团、数据所属外部系统（发送方）、单据类型等。这部分数据可以在 Servlet 的 URL 地址的参数中指定，也可以在 XML 数据文件的文档头属性中指定，但是在 Servlet 的 URL 地址的参数中指定的信息比在 XML 数据文件的文档头属性中指定的信息有更高的优先级别。比如在 Servlet 的 URL 地址参数中指定 groupcode=yk，而在 XML 数据文件的文档头属性中指定 groupcode=yy，则最终还是将数据发送到 yk 组织。

下面我们依据实例来逐一介绍各个常用属性。图 2.2.1 是一个典型的 NC 服务器的 Servlet 的 URL 地址，“http://10.7.3.226:80/service/XChangeServlet”是后台 Servlet 的服务名。“?”号之后的“account=0001&groupcode=xx&orgcode=yy”就是设置的 URL 地址参数，一般情况下这个地方只设置帐套编码(account)、接收集团(groupcode)，接收组织(orgcode)。至于这些属性的含义下面会介绍到。

http://10.7.3.226:80/service/XChangeServlet?account=0001&groupcode=xx&orgcode=yy

图 2.2.1 典型 Servlet 的 URL 地址

图 2.2.2 是一个典型的 XML 文档头，文档头的根标签名是 ufinterface，后面跟着就是各种初始化信息的设置。我们将这些属性分为三类：必须设置的属性、根据 Servlet 的 URL 参数设置决定是否必设的属性和根据需要决定是否设置的属性。

```
<ufinterface account="develop" billtype="user" groupcode="uap60" orgcode="uap60" filename="" isexchange="Y" replace="Y" roottag="" sender="001">
```

图 2.2.2 典型 XML 文档头

【必须设置的属性】:

billtype 属性，这个属性值决定了这个 XML 文件（文档）中所有单据的单据类型，信息交换平台所有的处理都是围绕单据类型的。

sender 属性，这个属性值设置的是外系统编码，指定的是数据的来源系统，即习惯上所说的发送方。

【根据 Servlet 的 URL 参数设置决定是否必设的属性】:

这些属性其实也是必设属性，但如果 Servlet 的 URL 参数里面设置了这几个属性的话，XML 文档头里面可以不设置这几个属性值，即使设置了也不会起作用。

account 属性,这个属性指定要将数据导入至 NC 系统的哪个帐套。

groupcode 属性，这个属性值指定接收集团的编码。

orgcode 属性，这个属性值指定接收组织的编码。

【根据需要决定是否设置的属性】:

filename 属性，在数据导入过程中，对于每张单据我们都可以记录其原始数据、翻译后数据，对整个文档我们也可以记录其导入后的回执信息，这个 filename 属性的值就是用于记录上述数据文件时的文件名。当然，如果您没有设置的话，系统会为每个导入的文档默认生成文件名。

isexchange 属性，这个属性值决定了在外系统的数据在导入 NC 系统的过程中，是否使用信息交换平台提供的翻译转换和校验功能。正常情况下应将这个属性设置为“Y”，或者干脆不设。除非从其他 NC 系统产生的符合 NC 转换后标准的 XML 数据直接导入 NC 系统，并且很多基础档案数据字段直接用的是 PK 值，此时可以设置属性为“N”和“n”，可避免无谓的翻译转换。

replace 属性，这个属性值决定是否允许将相同单据往同一个接收方重复导入。V50 版的插件一般允许相同单据重复导入，除了将第一次导入视作新增之外，其余导入视作更新。关于这方面的具体内容请参见 2.6 节内容。如果不允许相同单据的重复导入，那么将这个属性值设置为“N”或者“n”。否则将其设置为“Y”或者干脆不设。

operator 属性， 这个属性指定当前发送方对应的 nc 操作员 pk，在手工加载界面会自动使用当前登陆用户 pk，无需设置。对于外系统发送到 NC 系统，需要在其发送 url 里加上对应得 NC 操作员 pk

总的来说，XML 文档头的【必须设置的属性】和【根据 Servlet 的 URL 参数设置决定是否必设的属性】,均为在外系统交换文档发送过程中必须得到的初始化信息,可以在 Servlet 的 URL 地址参数中设置,也可以在 XML 文档头属性中设置,但必须进行设置。而对于 XML 文档头的【根据需要决定是否设置的属性】，也可以在 Servlet 的 URL 地址的参数中设置，但一般不推荐这么做。

2.3. 向 NC 系统发送数据方式

2.3.1. 手动界面发送

这是外部数据最简单也最常用的一种客户端发送方式，点击：[集成平台]-[数据交换管理]-[手动加载界面]，在打开的界面中选择需要发送的文件，如有必要，修改目标 URL 地址的 groupcode 参数值，选择“发送”菜单即可。详细内容请参考信息交换平台的用户手册。

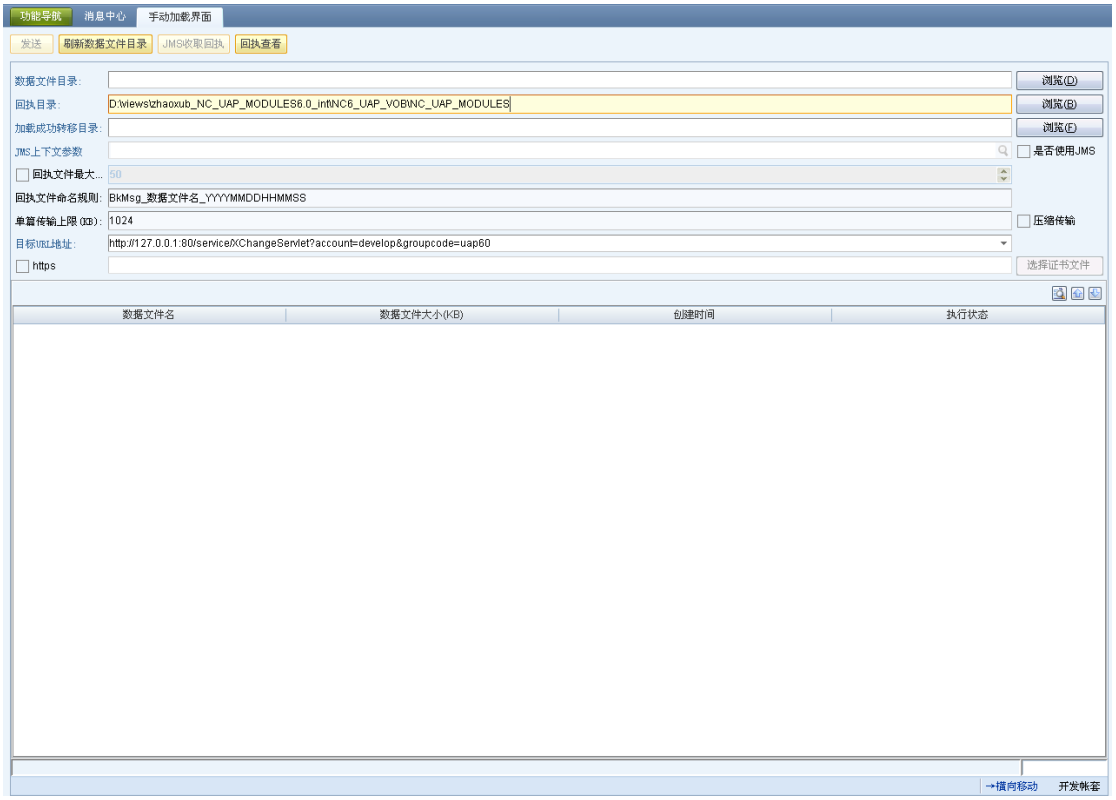
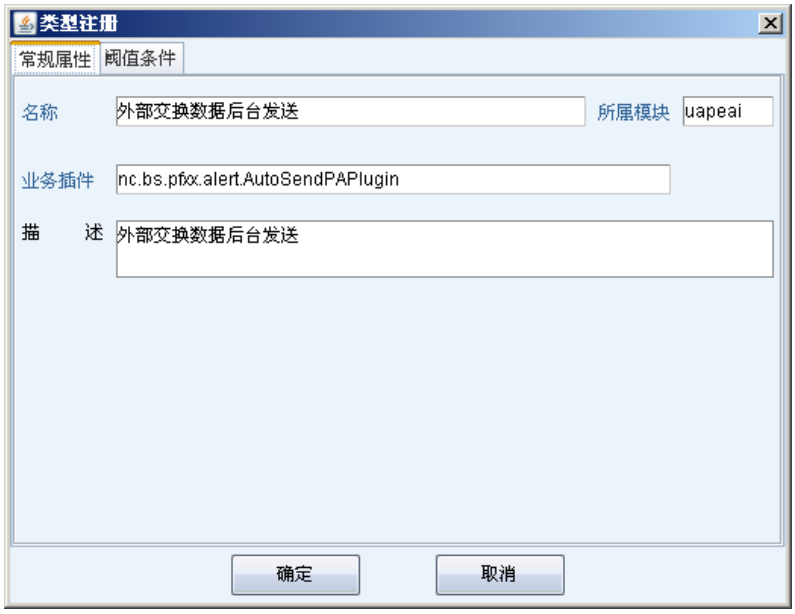


图 2.3.1 手动加载界面

2.3.2. 后台预警发送

点击：[开发平台]→[开发配置工具]→[后台任务类型注册]→ [增加]，在弹出的后台任务类型注册对话框中配置一个类型为“外部交换数据后台发送”的后台任务条目。



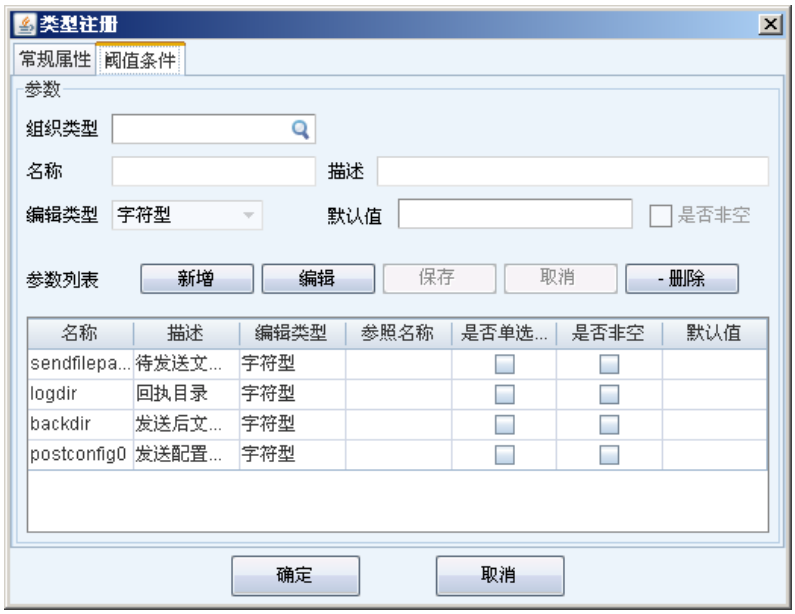


图 2.3.2 后台任务类型注册

其预警条件的配置如图 2.3.2 所示。在该预警条件标签页中内置了四个预警条件：

待发送文件目录

该预警条件阈值用于设置用户数据文件目录（不能为空）；

回执目录

该预警条件阈值用于设置存放单据导入之后的回执信息的目录，每张单据一个回执文件，如果用户不设置这个阈值，系统在待发送文件目录下默认创建目录/logdir 作为回执目录；

发送后文件转移目录

该预警条件阈值对于想将能够成功加载的文件备份走的用户有用（一般不用设置）；

发送配置信息

该预警条件阈值是一个可扩展的预警条件阈值，它用于配置将什么样的文件发送到哪个 URL 地址（不能为空）。这个配置值的格式如下：

第一部分是单据类型：

billtype=Test1;sysno= 1101;urladdress=http://localhost:80/service/XChangeServlet?account=0001&receiver=yy

第二部分是外系统编码：

billtype=Test1;sysno= 1101;urladdress=http://localhost:80/service/XChangeServlet?account=0001&receiver=yy

第三部分是目的 URL 地址：

billtype=Test1;sysno= 1101;urladdress=http://localhost:80/service/XChangeServlet?account=0001&receiver=yy

上面所列发送配置信息项取值的意思就是将单据类型是“Test1”， 外部系统编码为“1101”的文件发送到 NC 外部数据接收的 Servlet 地址：urladdress=http://localhost:80/service/XChangeServlet?account=0001&groupcode=zz。这个地方如果我们将单据类型的取值设为“XX”或“xx”的话，表示任何单据类型的文件都可以往某个地址发送，相当于通配符的意思，但对于外部系统编码没有通配符。

后台任务配置：

[企业建模平台]→[系统平台]→[后台任务中心] →[后台任务部署]中点击增加按钮

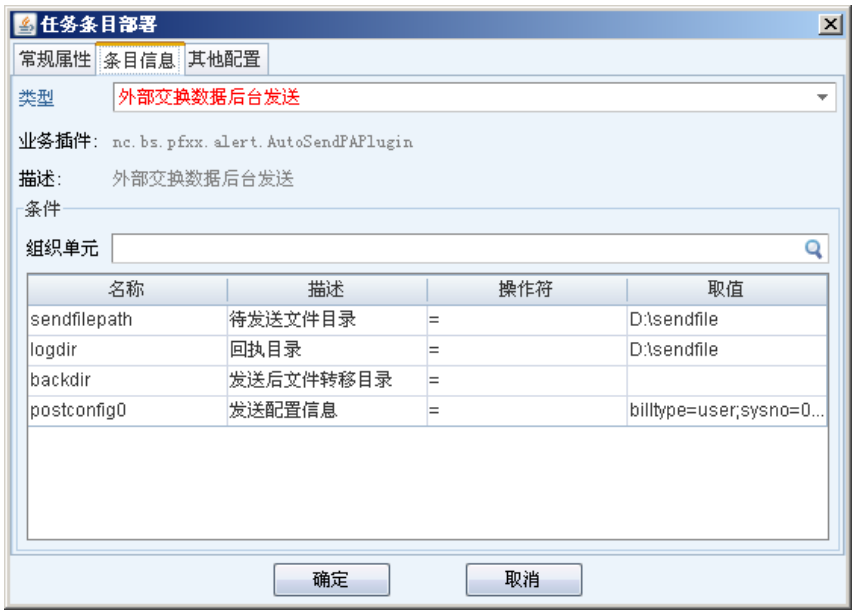


图 2.3.3 任务条目部署

发送配置信息是一个可扩展的预警条件阈值，其意思就是用户可以在预警平台的类型注册对话框中修改名为“外部交换数据后台发送”的预警类型定义，增加作为预警条件的发送配置信息项，以实现数据文件的多点发送。点击：[开发平台]→[开发配置工具]→[后台任务类型注册]→[选中“外部数据交换后台发送”行]→[修改]，弹出“类型注册”对话框，如图 2.3.4 所示，为其增加了一个发送配置信息阈值项。需要注意的是该新增阈值项的名称必须以字符串“postconfig”打头，如“postconfig1”、“postconfig2”，……等。这样配置更多的发送配置信息，使得能有选择地将某个特殊单据类型特殊外部系统的数据文件能够发送到某个指定服务器地址去。

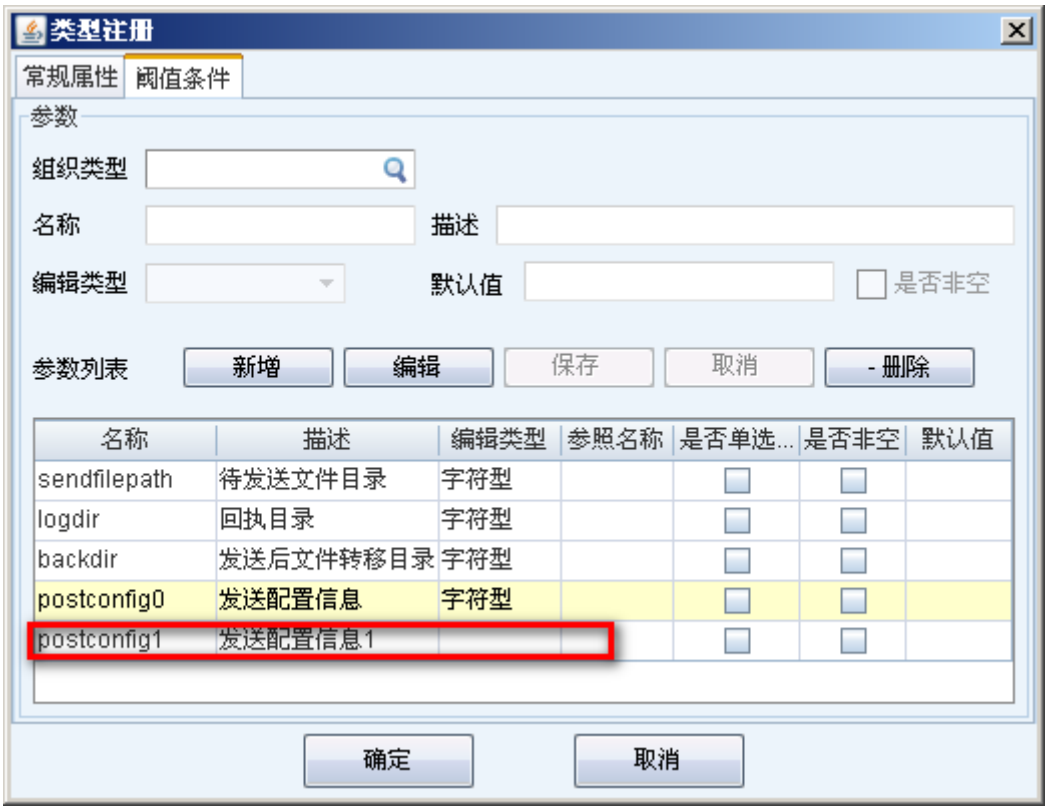


图 2.3.4 后台任务类型注册

配置完毕，即可由预警平台调度和执行发送任务。另请注意，在待发送文件目录、回执目录和发送后转移目录的设置上本地客户端和服务器端文件系统的差别，如目录“c:/aaa”指的是服务器端文件系统路径而非客户端文件系统地址。

2.3.3. 自定义程序发送

以 Java 代码为例，介绍外系统作为客户端如何向 NC 系统发送数据：

为简单起见，以下代码未处理异常，close 方法也未放入 finally 里，具体写代码时请修改。

```
        // 获取 Servlet 连接并设置请求的方法
        String url = "http://10.7.3.225:8080/service/XChangeServlet";
        URL realURL = new URL(url);
        HttpURLConnection connection =
(HttpURLConnection)realURL.openConnection();
```

```
            connection.setDoOutput(true);
            connection.setRequestProperty("Content-type", "text/xml");
            connection.setRequestMethod("Post");
            // 将 Document 对象写入连接的输出流中
            File file = new File("C:/samples/psndoc.xml");
            BufferedOutputStream out = new
BufferedOutputStream(connection.getOutputStream());
            BufferedInputStream input = new BufferedInputStream(new
FileInputStream(file));
            int length;
            byte[] buffer = new byte[1000];
            while ((length = input.read(buffer, 0, 1000)) != -1) {
                  out.write(buffer, 0, length);
            }
            input.close();
            out.close();
            // 从连接的输入流中取得回执信息
            InputStream inputStream = connection.getInputStream();
            Document resDoc = XMLUtil.getDocumentBuilder().parse(inputStream);    // 解
析为 Doc 对象
      // 对回执结果的后续处理…


      也可以调用 nc.vo.pfxx.pub.PostFile 来发送
      public static int sendFile(File file, String url, String backdir,
                  String movedir, boolean bcompress)
      // fileQueue 传空即可
      public static SendResult sendFileWithResults(File file, String url, String backdir, String
movedir, boolean bcompress, FileQueue fileQueue)
```

public static String sendDocument(Document doc, String url,String outputEncoding, boolean bcompress) throws Exception

public static String sendDocument (Document doc, String url, String outputEncoding) throws Exception

[说明] 关于 XML 操作可以用 XMLUtil 类,关于发送的 API，如果您的程序可以依赖 uap 的 jar 包，也可以直接用 PostFile 类

2.4. 回执及异常出错信息

2.4.1. 回执格式

加载外部系统数据时，对于导入的数据文件中的每张单据，都会给客户端返回一个回执信息，记录单据导入成功与否，以及失败的原因。回执信息也是一个 XML 文件，其一般格式如下：

```
<?xml version="1.0" encoding='UTF-8'?>
<ufinterface billtype="bspsn" filename="人员档案样本数据文件.xml" isexchange="Y" receiver="yk" replace="Y" roottag="sendresult" sender="1001" successful="N">
    <sendresult>
        <billpk>
        </billpk>
        <bdocid>12315601022323</bdocid>
        <filename>
        </filename>
        <resultcode>-31202</resultcode>
        <resultdescription>单据 12315601022323 开始处理...
```

(1)单据翻译转换错误:根据基础档案[部门档案]无法翻译[pk_deptdoc]字段,待翻译值:部门 yk,翻译方式:按名称., 翻译参照的公司：yk。

(2)单据翻译转换错误:根据基础档案[岗位档案]无法翻译[pk_om_job]字段,待翻译值:测试,翻译方式:按名称.，翻译参照的公司：yk。

(3)单据翻译转换错误:根据基础档案[人员类别]无法翻译[pk_psncl]字段,待翻译值:测试人员 yk,翻译方式:按名称.，翻译参照的公司：yk。

```
单据 12315601022323 处理错误:单据翻译转换错误:单据校验转换失败
</resultdescription>
        <content>
        </content>
        <invaliddoc>
            <docitem name="abc">部门 yk</docitem>
            <docitem name="岗位档案">测试</docitem>
            <docitem name="人员类别">测试人员 yk</docitem>
        </invaliddoc>
    </sendresult>
</ufinterface>
```

图 2.4.1 回执文档的一般格式

回执信息中还可以嵌套单据，这个时候可以将信息交换平台看作简单的信息中心，外系统发送过来一个消息，NC 系统回应一个消息。如查询类的单据，外系统发送一个表示查询条件的信息，NC 系统返回所有符合查询条件的单据，这些单据就是内嵌在回执信息中的。再如，用这个功能来实现 NC 系统和外系统在某种程度上的流程集成，外系统发送一张走流程平台的单据，然后再由信息交换平台返回流程平台处理后得到的单据。

2.4.2. 异常和错误编码

异常信息对于使用信息交换平台是极为重要的，有助于实施人员分析数据导入结果，查找数据导入失败原因。信息交换平台中的异常分类：环境初始化异常、单据转换异常和业务插件处理异常。这个划分是基于一个外系统某单据类型的 XML 交换文档在导入 NC 系统过程中所需处理阶段而给出的。如果单据导入失败，那么在回执信息的<resultdescription>中会

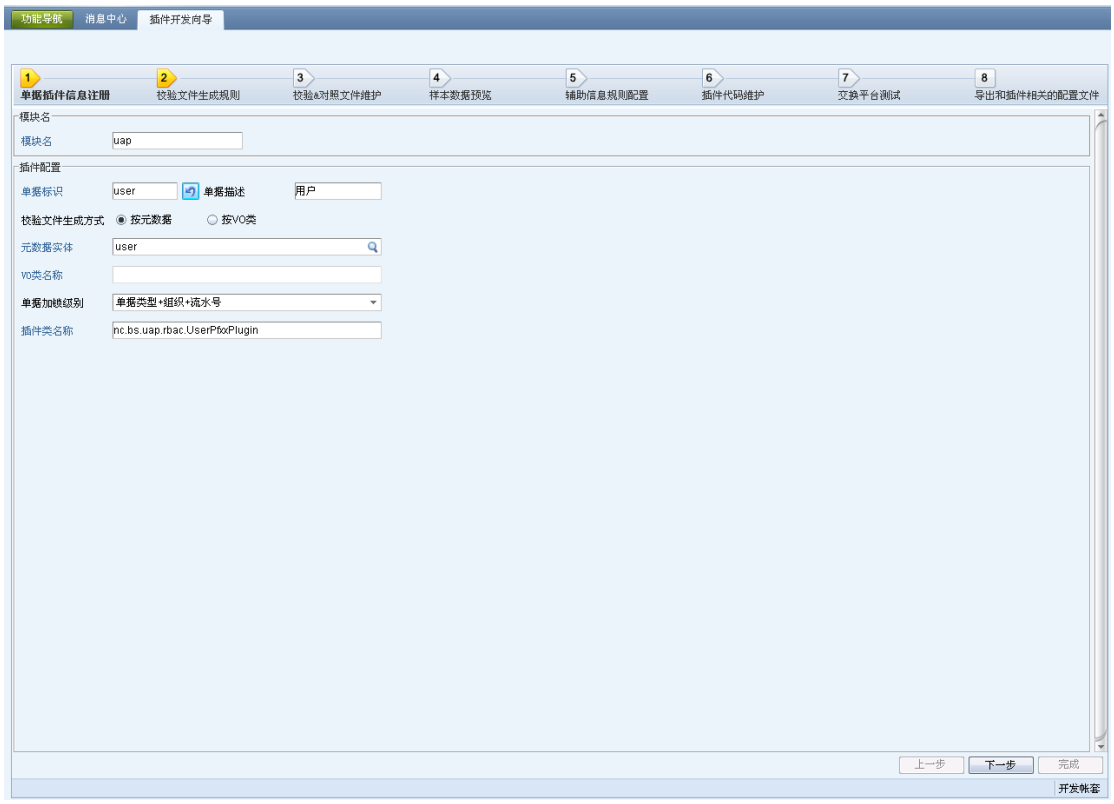
给出发生了什么异常以及出错信息。如果需要更详细的出错信息，请检查后台日志，信息交换平台对于导入数据过程中发生的异常，记录在 nchome/nclogs/server/pfxx-log.log 日志文件中。

错误编码则直接定位于发生错误的分类：配置错误、数据错误、系统错误，然后在某种错误下再进行细分。这些错误信息都通过错误编码的方式在单据导入的回执信息的<resultcode>中给出。这些错误编码信息主要是给第三方系统程序内部用于识别数据交换过程中是否出现问题，以及具体出了什么问题，便于程序内进行控制。对于实施人员，如果拿到一份回执，只需要知道 1 表示正确传送，其他的代表各种错误码，一般均为负数。

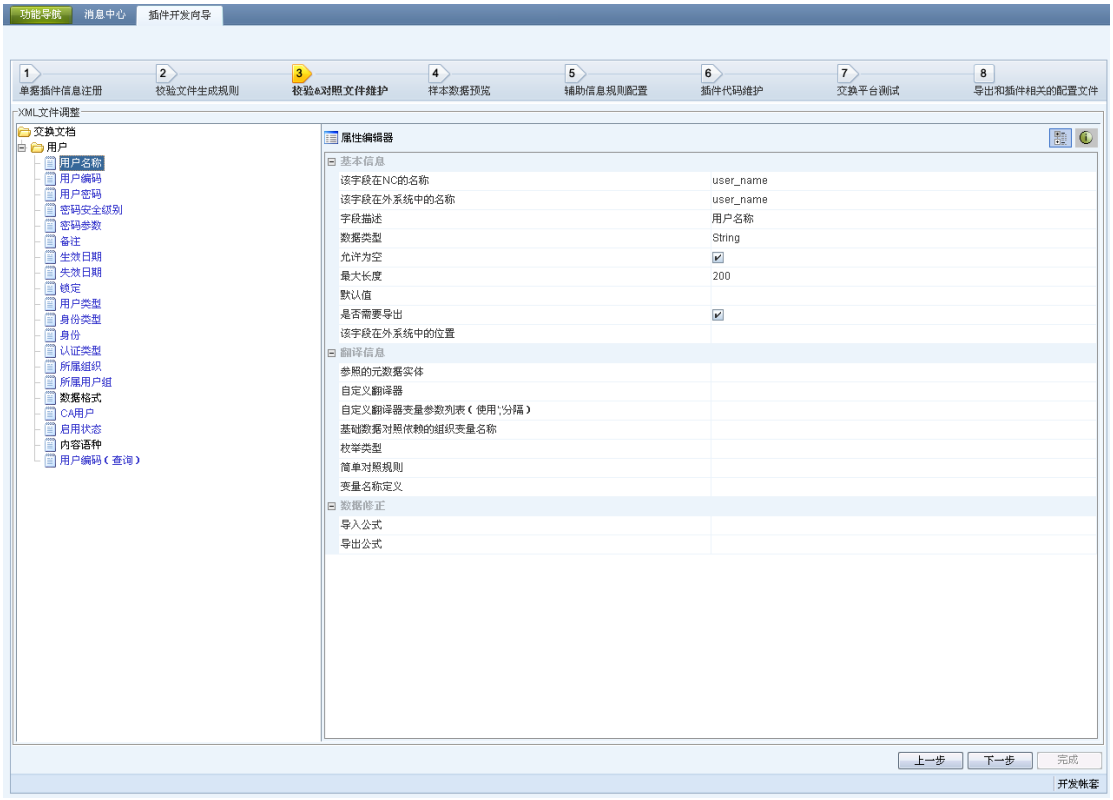
对于无法翻译的文档，在回执中也有具体描述，其中<invaliddoc></invaliddoc>段中<docitem name="abc">部门 yk</docitem> name 表示改文档在外系统中对应的文档名称(如果 nc 基本档案资源列表里没有，则为基础数据对照表中对应外系统档案名称)，文本内容为其代翻译的值。

2.4.3. 查询及回执格式样例

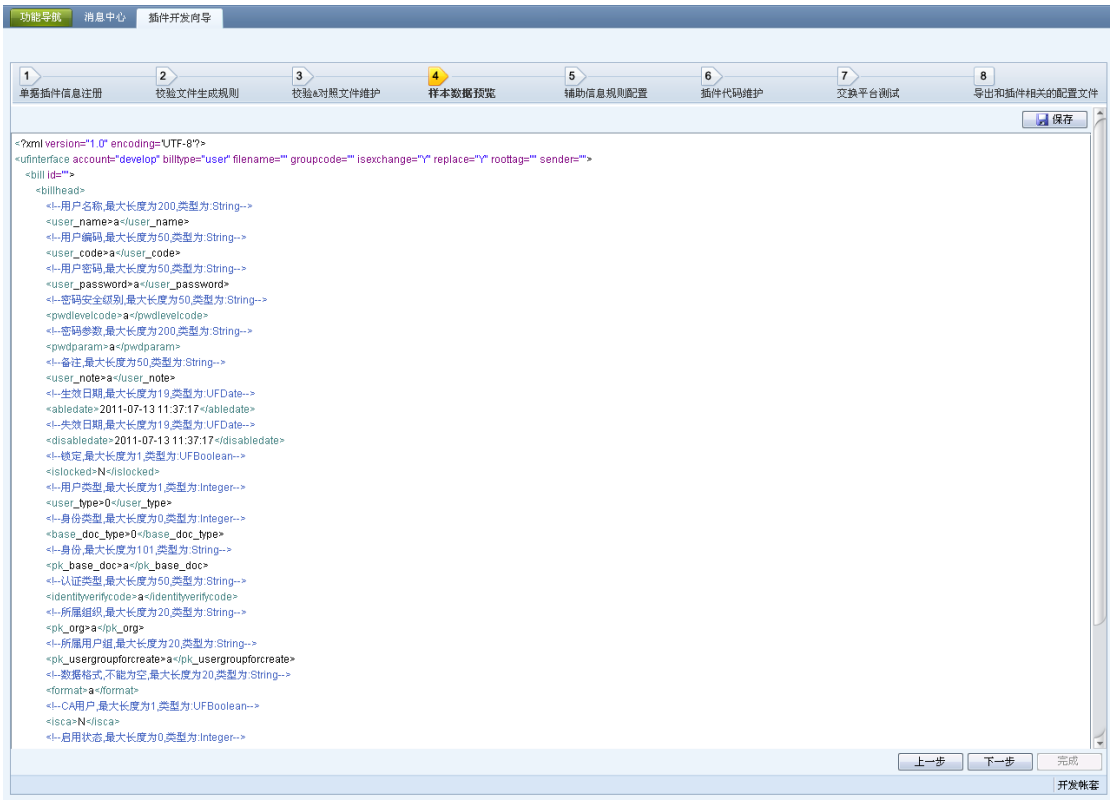
注册查询单据类型



生成交换规则



生成样本文件并保存为文件



修改样例

```
<?xml version="1.0" encoding='UTF-8'?>
```

```
<ufinterface account="develop" billtype="user" groupcode="" orgcode="" filename=""
isexchange="Y" replace="Y" roottag="" sender="">
    <bill id="">
        <billhead>
            <!--用户名称,最大长度为 200,类型为:String-->
            <user_name>a</user_name>
            <!--用户编码,最大长度为 50,类型为:String-->
            <user_code>a</user_code>
            <!--用户密码,最大长度为 50,类型为:String-->
            <user_password>a</user_password>
            <!--密码安全级别,最大长度为 50,类型为:String-->
            <pwdlevelcode>a</pwdlevelcode>
            <!--密码参数,最大长度为 200,类型为:String-->
            <pwdparam>a</pwdparam>
            <!--备注,最大长度为 50,类型为:String-->
            <user_note>a</user_note>
            <!--生效日期,最大长度为 19,类型为:UFDate-->
            <abledate>2011-07-13 11:37:17</abledate>
            <!--失效日期,最大长度为 19,类型为:UFDate-->
            <disabledate>2011-07-13 11:37:17</disabledate>
            <!--锁定,最大长度为 1,类型为:UFBoolean-->
            <islocked>N</islocked>
            <!--用户类型,最大长度为 1,类型为:Integer-->
            <user_type>0</user_type>
            <!--身份类型,最大长度为 0,类型为:Integer-->
            <base_doc_type>0</base_doc_type>
            <!--身份,最大长度为 101,类型为:String-->
            <pk_base_doc>a</pk_base_doc>
            <!--认证类型,最大长度为 50,类型为:String-->
```

```
                <identityverifycode>a</identityverifycode>
                <!--所属组织,最大长度为 20,类型为:String-->
                <pk_org>a</pk_org>
                <!--所属用户组,最大长度为 20,类型为:String-->
                <pk_usergroupforcreate>a</pk_usergroupforcreate>
                <!--数据格式,不能为空,最大长度为 20,类型为:String-->
                <format>a</format>
                <!--CA 用户,最大长度为 1,类型为:UFBoolean-->
                <isca>N</isca>
                <!--启用状态,最大长度为 0,类型为:Integer-->
                <enablestate>0</enablestate>
                <!--内容语种,不能为空,最大长度为 20,类型为:String-->
                <contentlang>a</contentlang>
                <!--用户编码（查询）,最大长度为 50,类型为:String-->
                <user_code_q>a</user_code_q>
            </billhead>
        </bill>
</ufinterface>
```

插件样例

```
package pfxx.example11;


import nc.bs.framework.common.NCLocator;
import nc.bs.logging.Logger;
import nc.bs.pfxx.ISwapContext;
import nc.itf.uap.bd.accsubj.IAccsubjDataQuery;
import nc.itf.uap.pfxx.IPFxxEJBService;
import nc.vo.bd.b02.AccsubjVO;
import nc.vo.pfxx.auxiliary.AggxsysregisterVO;
```

```
import nc.vo.pfxx.exception.PfxxPluginException;
import nc.vo.pfxx.util.PfxxPluginUtils;
import nc.vo.pfxx.util.PfxxUtils;
import nc.vo.pub.BusinessException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/**
 * <b> 在此处简要描述此类的功能 </b>
 *
 * <p>
 *  在此处添加此类的描述信息
 * </p>
 *
 * Create on 2006-4-6 16:00:51
 *
 * @author ufsoft
 * @version Your Project Ver5.0 @since V5
 */
public class QueryPlugin extends nc.bs.pfxx.plugin.AbstractPfxxPlugin {

    /**
     * 将由 XML 转换过来的 VO 导入 NC 系统。业务插件实现此方法即可。<br>
     * 请注意，业务方法的校验一定要充分
     *
     * @param vo
     *                转换后的 vo 数据，在 NC 系统中可能为
```

ValueObject,SuperVO,AggregatedValueObject,IExAggVO 等

```
        *                    。
        * @param swapContext
        *                    各种交换参数，公司，帐簿，接受方，发送方，主体帐簿等等
        * @param aggvo
        *                    辅助信息 vo
        * @return
        * @throws BusinessException
        * @date 2006-1-11
        */
       protected Object processBill(Object vo, ISwapContext swapContext,
AggxsysregisterVO aggvo) throws BusinessException {

            // 1.得到转换后的 VO 数据,取决于向导第一步注册的 VO 信息
            CommonQueryVO resvo = (CommonQueryVO) vo;
            KeyValueVO[] keyValueVOs = resvo.getConditions();
            int n = keyValueVOs.length;
            String[] keys = new String[n];
            for (int i = 0; i < n; i++) {
                keys[i] = keyValueVOs[i].getValue();
            }

            // 2.查询此单据是否已经被导入过，有三个方法，具体使用哪一个请参考方法
说明 javadoc
            // 1) String vopk = PfxxPluginUtils.queryBillPKBeforeSaveOrUpdate(ufvo);
            // 2) String vopk =
            // PfxxPluginUtils.queryBillPKBeforeSaveOrUpdate(ufvo,resvo.getPk_corp);
            // 3) String vopk =
            // PfxxPluginUtils.queryBillPKBeforeSaveOrUpdate(ufvo,resvo
```

```
            // .getPk_corp(),resvo.getPk_glogBook());

            // 3. 如果单据设置有辅助信息，xsysvo 为用户配置的具体辅助信息

            // 4.如果此单据没有导入过，那么准备保存新单据，保存单据前请进行必要的
数据检查，并给出明确的业务异常...
            // TODO
//          Logger.info("如果此单据没有导入过,保存新单据...");

            // 5.如果此单据已经导入过，请调用
PfxxPluginUtils.checkBillCanBeUpdate(UfinterfaceVO
            // ufvo)检查单据是否允许更新
            //  如果不允许更新,此方法会抛出业务异常
            // TODO
//          Logger.info("如果单据已经导入过则进行单据更新...");

            // !!!!测试转换后的单据数据，正式代码里请删除
            // Document doc = XMLUtil.getNewDocument();
            // new com.thoughtworks.xstream.XStream().toXML(resvo,doc);
            // try
            // {
            // FileUtils.writeDocToXMLFile(doc,
            // PfxxServerSidePathVocabulary.EXPORTBILLS_PATH+
            // "test/nc/uap/pfxx/kjkm/QueryPlugin.xml");
            // } catch (java.io.IOException e)
            // {
            // e.printStackTrace();
            // }
            // !!!!测试代码结束
```

```
            // !!!!查询 NC 数据，通过回执返回到外系统示例代码开始
            Document resdoc = null;
            /*  获取导出服务  */
            IPFxxEJBService runner = PfxxUtils.lookUpPFxxEJBService();
            try {

                /*  查询需要导出的 VO,这里查全部,实际应用应该通过 XML 传递条件过来进行查询  */
                IAccsubjDataQuery queryService = NCLocator.getInstance().lookup(IAccsubjDataQuery.class);
                AccsubjVO[] vos = queryService.queryAccsubjVosByPks(keys);

                // Collection col = (new
                // nc.bs.dao.BaseDAO()).retrieveByClause(CommonQueryVO.class,"1=1");
                /*  实例代码仅演示 ValueObject,如果是 AGGVO 请修改  */
                resdoc = runner.exportBills(vos, swapContext.getAccount(), "bsaccsubj", swapContext.getOrgPk(), swapContext.getSender(), null);

                //get ids from document and insert to db
                String[] docids = new String[vos.length];
                NodeList basdocList=resdoc.getDocumentElement().getElementsByTagName("basdoc");

                int total=basdocList.getLength();
                for(int i=0;i<total;i++)
                {
                    Element node=(Element)basdocList.item(i);
                    docids[i]=node.getAttribute("id");
```

```
                }
                for (int i = 0; i < docids.length; i++) {
                    PfxxPluginUtils.addDocIDVsPKContrast("bsaccsubj",docids[i],
swapContext.getOrgPk(), vos[i].getPk_accsubj());
                }

//                StringBuffer sbRet=new StringBuffer();
//                XMLUtil.writeXMLFormatString(sbRet, resdoc, 0);
//                System.out.println();
            } catch (Exception e) {
                Logger.error("导出单据失败", e);
                throw new PfxxPluginException(this.getClass().getName(), "导出单据失败:"
+ e.getMessage());
            }
            // !!!!查询 NC 数据，通过回执返回到外系统示例代码结束
            // 6.如果希望单据将来可以更新，请调用下列接口插入文档流水号与生成 PK
的对照关系
            // 第 2 步查询对照关系相对应，也有三个方法，具体请看 javadoc
            // 1)PfxxPluginUtils.addDocIDVsPKContrast(ufvo,pk);
            // 2)PfxxPluginUtils.addDocIDVsPKContrast(ufvo,resvo.getPk_corp,pk);
            // 3)PfxxPluginUtils.addDocIDVsPKContrast(ufvo,resvo.getPk_corp(),resvo.
            // getPk_glogBook(),pk);
            // 7.准备返回值,此函数的返回值，最终会以字符串的形式返回给外系统，
            // 对于普通单据可以返回 NC 系统生成的 PK 值，对于凭证可能返回凭证号，
具体视单据而定
            // 对于查询插件要求返回 org.w3c.dom.Node[]数组 或者 org.w3c.dom.Node
            return resdoc;
```

```
    }
}
```

回执结果

```
<?xml version="1.0" encoding='UTF-8'?>
<ufinterface billtype="kjkm1" filename="kjkm1e9a8164U860.xml" isexchange="Y" receiver="0001" replace="Y" roottag="sendresult" sender="U860" successful="Y">
    <sendresult>
        <billpk>
        </billpk>
        <bdocid>CORPe40702bkjkmquery</bdocid>
        <filename>kjkm1e9a8164U860.xml</filename>
        <resultcode>1</resultcode>
        <resultdescription>单据 CORPe40702bkjkmquery 开始处理...单据 CORPe40702bkjkmquery 处理完毕!</resultdescription>
        <content>
        </content>
    </sendresult>
  <ufinterface>
        <ufinterface account="design" basedocid="0001a9f7bbsaccsubj" billtype="bsaccsubj" filename="0001a9f7bbsaccsubj" isexchange="Y" proc="" receiver="U860" replace="Y" roottag="code" sender="0001" subbilltype="">
            <code id="0001a9f7bbsaccsubj0">
                <balanflag>N</balanflag>
                <prop>True</prop>
                <beginPeriod>01</beginPeriod>
                <beginYear>2007</beginYear>
                <bothorient>Y</bothorient>
                <cashbankflag>0</cashbankflag>
                <createPeriod>01</createPeriod>
```

<createYear>2007</createYear>

<createcorp>0001</createcorp>

<ctlsystem>

</ctlsystem>

<innersubj>N</innersubj>

<accremove>N</accremove>

<fc_name>人民币</fc_name>

<dispname>应收利息</dispname>

<endPeriod>

</endPeriod>

<ename>

</ename>

<free1>

</free1>

<free2>

</free2>

<free3>

</free3>

<free4>

</free4>

<free5>

</free5>

<free6>

</free6>

<free7>

</free7>

<free8>

</free8>

<free9>

```
</free9>
<grpaccsubjname>
</grpaccsubjname>
<incurflag>N</incurflag>
<innerinfonull>N</innerinfonull>
<outflag>N</outflag>
<pk_accsubj>0001111000000000009RI</pk_accsubj>
<pk_corp>0001</pk_corp>
<pk_grpaccsubj>
</pk_grpaccsubj>
<type>资产</type>
<property1>
</property1>
<property2>
</property2>
<property3>
</property3>
<property4>
</property4>
<property5>
</property5>
<remcode>
</remcode>
<acc_seal_flag>
</acc_seal_flag>
<person_acc>false</person_acc>
<code>1132</code>
<name>应收利息</name>
<cust_acc>false</cust_acc>
```

```
                    <supplier_acc>false</supplier_acc>
                    <dept_acc>false</dept_acc>
                    <item_category>客商辅助核算</item_category>
                    <item_acc>true</item_acc>
                    <subjlev>1</subjlev>
                    <measure_unit>
                    </measure_unit>
                    <pk_glorgbook>1-0002</pk_glorgbook>
                    <cash_acc_flag>
                    </cash_acc_flag>
                    <bank_acc_flag>
                    </bank_acc_flag>
                    <subjcontrol>
                    </subjcontrol>
                </code>
            </ufinterface>
        </ufinterface>
</ufinterface>
```

2.5. 信息交换平台总体参数设置

点击：[集成平台]-[数据交换管理]-[交换平台日志]-[参数设置]，在交换平台日志界面打开参数设置对话框，如图 2.5.1 所示。在这个对话框中，可以设置在外部数据导入过程中起到相当作用的参数，您可以根据需要对这些参数的设置进行改动。设置的参数值实时生效，不用重起后台服务器。

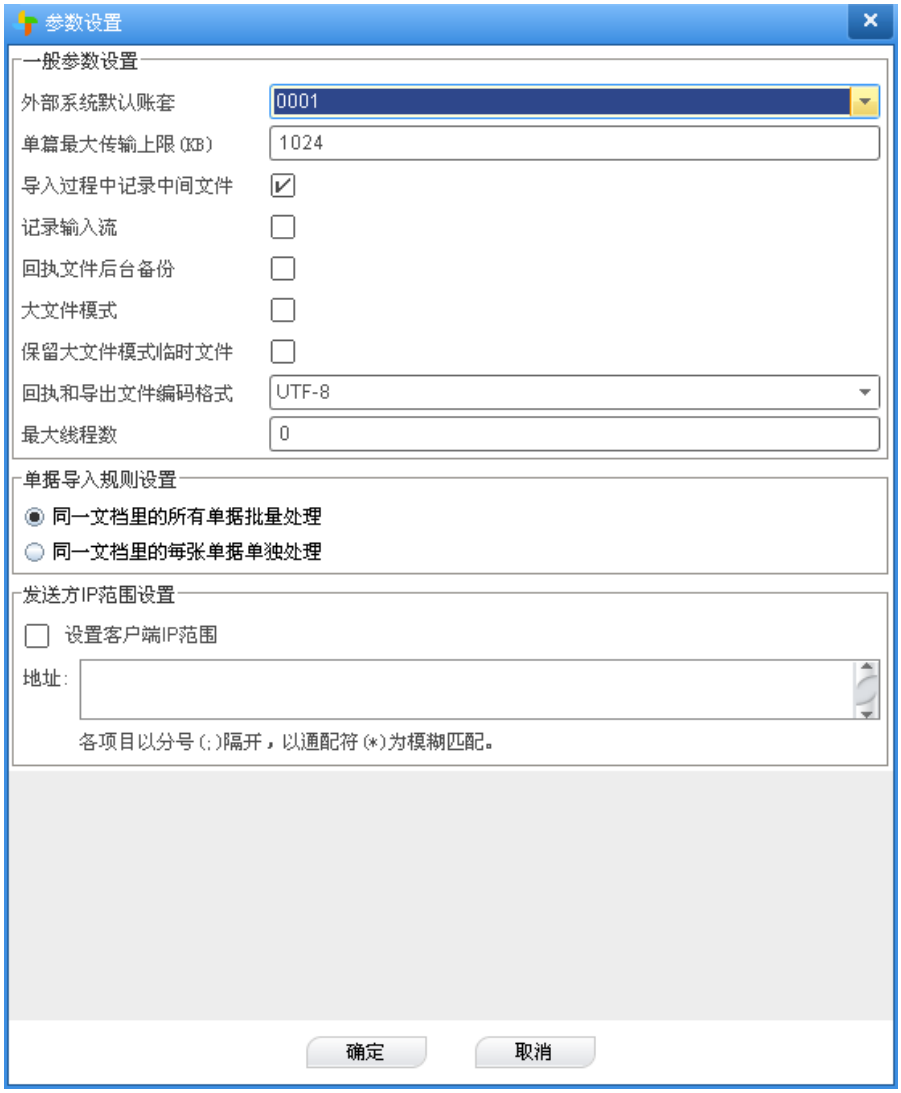


图 2.5.1 总体参数设置

下面我们来介绍这些参数的功能及需要注意的地方：

2.5.1. 外部系统默认帐套

将某个外系统数据文档导入到哪个帐套是由 Servlet 的 URL 地址的 account 参数或者文档根属性 account 决定的，但是如果这两个地方都没有申明这个值，那么系统就将外部数据导入至这个参数设置的帐套中。系统默认没有设置这个属性，但打开对话框就会显示当前系统所有帐套编码。

2.5.2. 单篇最大传输上限

对于要导入的单篇文档或者单个文件来说，其大小是受服务器端系统性能影响的。系统默认单篇最大传输上限是 1024KB，这个大小一般系统的存储器都能够满足。如果用户导入的文件大于设置的值，回执文件就会提示“文件长度超过设定长度”的错误。因此，用户可以根据需要，如待导入文件的大小和机器性能调整这个值，目前在界面可设置最大值为 10M；该值不建议设置过大，在 5000KB 以下，以便得到较好的传输性能。如确实需要传输超过 10M 文件，请自行在应用服务器上修改 NCHOME/pfxx/globalset.xml 中 max-transfer-size 的值，并考虑 OutOfMemory 的风险。

2.5.3. 导入过程是否记录中间文件

这个参数主要用于控制所导入文档的每张单据是否在服务器本地做备份记录，主要做两次备份，分别是原始单据备份和转换翻译后单据的备份，前者备份至目录 NC_HOME/pfxx/pfxxtemp/indocs/下，后者备份至目录 NC_HOME/pfxx/pfxxtemp/translated/下。这些信息在找错排错中是非常有用的，比如发生找不到某个字段元素的错误，它可能是交换规则定义错误引起，也可能是外系统数据格式错误引起，这个时候将备份的导入原始单据和转换后的单据做一个对比，便可以很快定位问题。系统默认是勾选“导入过程是否记录中间文件”。

2.5.4. 回执文件后台备份

对于导入的每一张单据，信息交换平台都会给客户端或者发送方一个回执信息即一个 XML 格式的回执文档。如果您使用的数据发送方式是手动界面发送或者后台预警发送的话，回执信息都会以文件的形式保存到指定目录下，但是如果通过自定义程序实时发送数据的话，当出现传输异常时就需要在后台备份一下单据的每个回执文件，目的就是检查返回给发送方的数据是否正确。回执文件后台备份这个参数就是用于控制是否后台备份这个文件，系统默认不进行备份。

2.5.5. 回执和导出文件编码格式

这也是一个便利工具，目的就是使得信息交换平台导出的 XML 数据文件或者单据回执文件的编码格式可配置。如果与 NC 系统交互的第三方系统只能处理 gb2312 编码格式的文件，经过配置之后就可以相当方便的达到目的，只是此时如果回执信息包含非 gb2312 编码字符，则可能显示不正常。系统默认回执和导出文件的编码格式是 UTF-8，信息交换平台同时支持 UTF-8 和 gb2312 格式文件的导入。

2.5.6. 单据导入规则设置

信息交换平台以交换文档中的单据为单位，支持非独立事务和独立事务两种导入方式，即同一 XML 文档对象中的所有单据批量处理和同一 XML 文档对象中的每张单据单独处理。例如，一个凭证类型的 XML 文档由一百张凭证组成，如果将交换平台总体参数的单据导入规则设置为“同一文档里的所有单据批量处理”的话，那么如果在导入这个文档的过程中有任何一张凭证发生错误，将导致所有的这一百张凭证导入失败，但是如果单据导入规则设置为“同一文档里的每张单据独立处理”的话，那么即使这一百张凭证里有些凭证在导入过程中失败了，也不会影响到其余单据的成功导入。通俗一点的说话就是，非独立事务就是文档中的所有单据要么一块儿成功，要么一块儿失败，而独立事务就是文档中的每张单据的成功导入与否不会受其他单据导入成功与否的影响，当然如果从文档中导入的单据有先后顺序的话，选择独立事务可能也就起不到作用了。系统默认的单据导入方式是“同一文档里的所有单据批量处理”。

2.5.7. 设置客户端 IP 范围

这个参数属于信息交换平台安全性方面的设置。我们在这里设置允许向 NC 系统发送数据文件的地址范围，这样我们可以确保与合法的外部系统进行数据集成。如，假设目前我们只允许地址为 10.7.3.255 和 10.7.3.233 两个 IP 地址的客户端发送数据，那么我们就启用这个属性设置，并将这两个 IP 地址输入至其输入框中，中间使用；号隔开。设置完这个属性之后，我们再去利用这两个之外的 IP 地址的客户端发送外系统数据的话，回执文件就会提示“发送方地址不合法!”。

2.6. 单据流水号和单据并发控制

2.6.1. 单据流水号的概念和作用

单据流水号在信息交换平台中是一个相当重要的概念，可以将其看作一张单据在信息交换平台中的标识。一般情况下，需要为待导入的每张单据定义一个流水号，如果未定义的话，系统会自动为导入的每张单据生成一个独一无二的流水号。信息交换平台将同一单据类型并具有相同流水号的单据看作是同一张单据，如果重复往同一接收方发送具有相同流水号的单据时，第一次会在 NC 系统新增该单据，其后均更新已导入单据。但如果往不同接收方导入具有相同流水号的单据，第一次同样是新增该单据，其后也是新增且是往不同接收方新增，但往往会发生编码名称重复等错误，取决于不同的档案或者单据在不同组织内是否控制编码或者名称重复。

我们可以将数据导入与界面录入做一比较：往某个接收方如 A 公司发送某种单据类型的数据，就相当于 A 公司某个用户登陆并打开相应单据界面进行制单。第一次往 A 公司发送某个流水号的单据，就相当于在界面新增一个单据；再一次往 A 公司发送具有相同流水号的单据，就相当于在界面上修改原来新增的单据一样。这个时候如果将具有相同流水号的单据发送至 B 公司的话，就相当于以 B 公司某个用户登陆系统，在相应单据界面上新增一张与原来在 A 公司新增的单据在数据上一模一样的单据。如果根据业务这在很多系统中是不允许的，信息交换平台跟前台界面操作一样，同样会报错。

2.6.2. 单据并发控制

点击：[集成平台]-[数据交换管理]-[集成工具开发]-[插件开发向导]，打开业务插件配置界面如下。

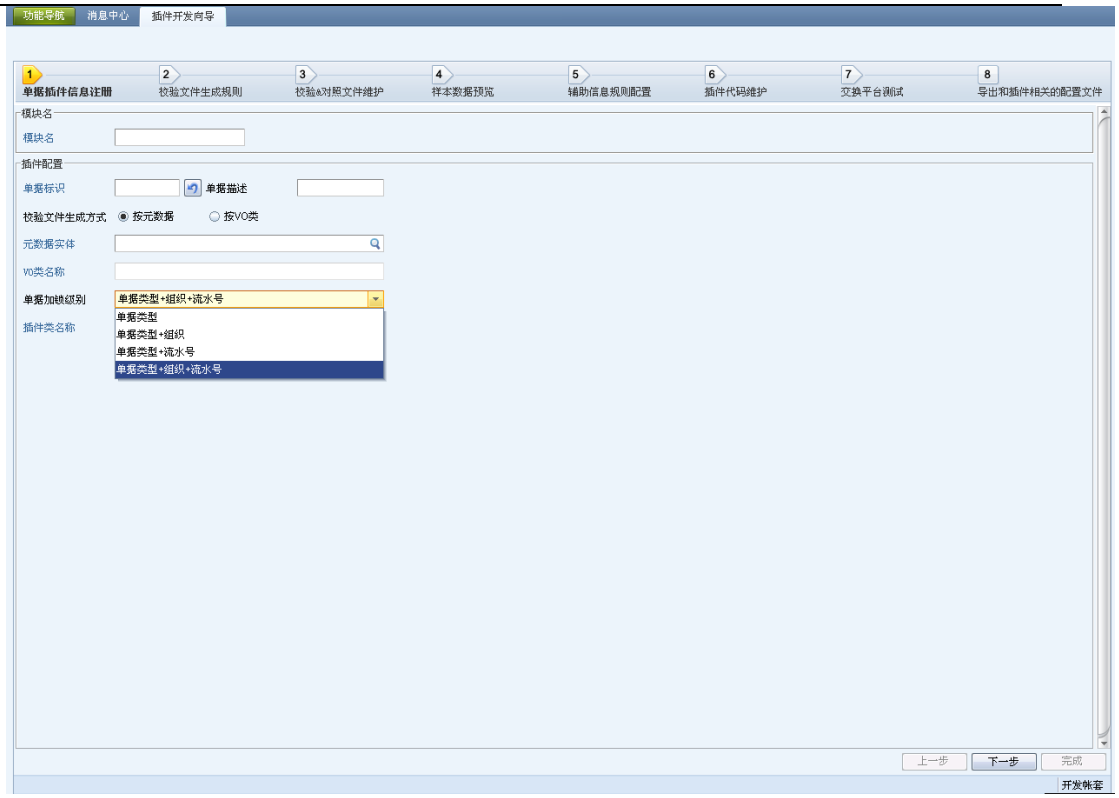


图 2.5.2 为业务单据设置并发级别

系统为每种单据类型的数据内置了四个并发加锁级别，由低到高分别是：单据类型、单据类型+组织、单据类型+流水号、单据类型+组织+流水号。

2.7. 日志查看

从 V55 开始，信息交换平台加强了前台日志查询功能，通过前台查看后台日志可以准确定位一些常见的错误和问题。

详细日志

单据类型　　发送文档单据类型

发送方　　外部系统

接收方　　接收组织

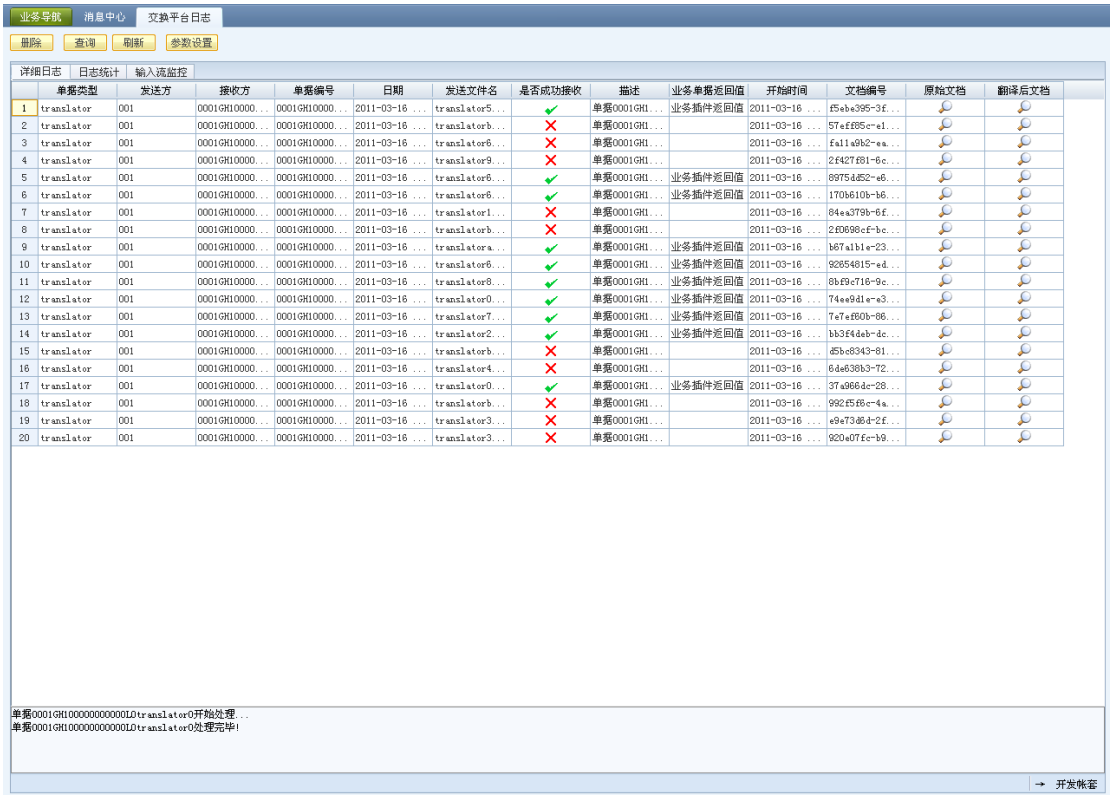
单据编号　　文档 id

是否成功接收　　红叉表示失败

业务单据返回值　业务插件返回值

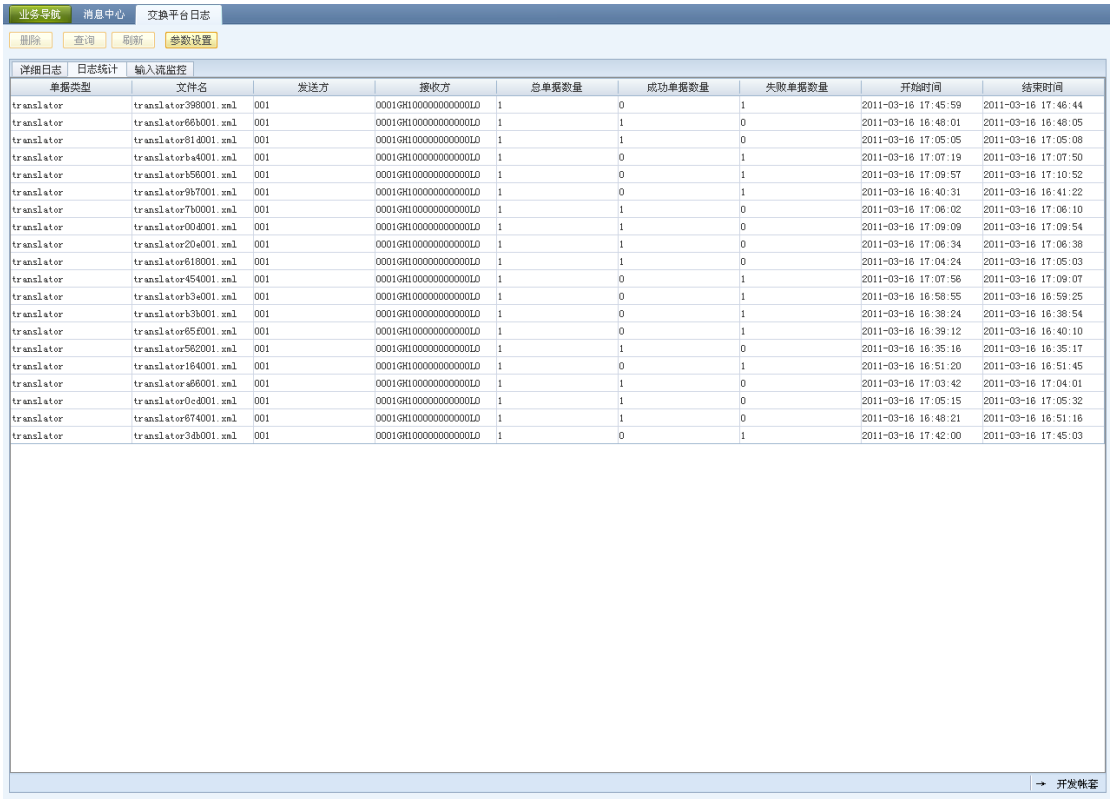
原始文档　　原始输入文档

翻译后文档　　交换平台翻译后文档



日志统计

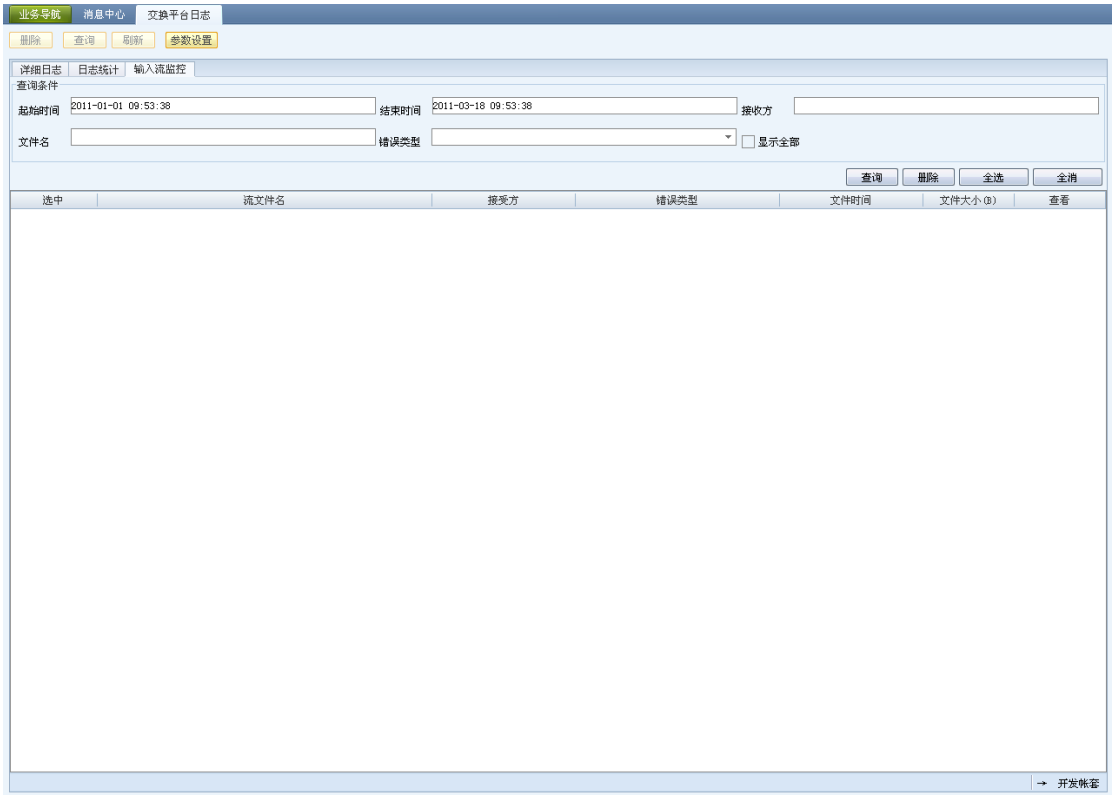
统计一次发送的文档中的单据总数量、成功数量、失败数量



输入流监控

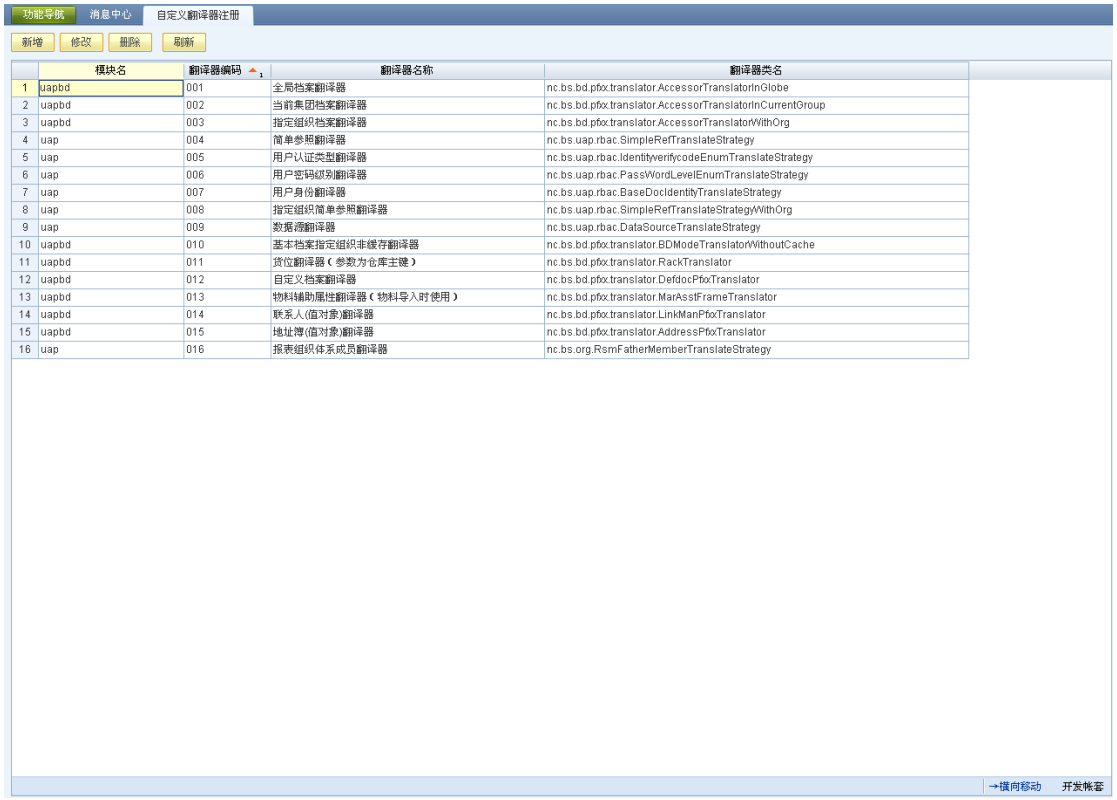
在往信息交换平台网络传输的过程中，不可避免地会出现网络传输错误、文档格式错误、文档翻译错误，和正常文档。因此准确定位错误类型，有助于快速解决问题。打开输入流监控，将降低系统性能。

在参数设置中打开输入流监控，这样所有通过信息交换平台传输的数据将会被记录。查看错误类型及查看出入文档原始文档



2.8. 翻译器配置

配置翻译器，可以根据用户需要创建个性化的翻译器。自定义翻译器需要实现 nc.bs.pfxx.ITranslatorStrategy 接口，并将新的翻译器类名注册到“自定义翻译器注册”界面



nc.bs.pfxx.ITranslatorStrategy 接口说明：

1.translateExToNC(String srcValue, String metaDataID, ITranslateContext translateContext)

将外部数据翻译为 NC 主键。

2. translateNCToEx(String docPk, String metaDataID, ITranslateContext translateContext)

将 NC 主键翻译为外系统能够识别的数据。

上述两个方法中都有上下文 ITranslateContext 作为参数，该上下文中提供了当前集团主键，组织主键，外系统主键，翻译策略（仅按对照表；按 PK；按编码；按名称）及用户自定义变量。

翻译器开发人员可以根据提供的上下文进行数据的翻译。

自定义变量的获取：

可以通过定义的变量的名称作为 key 取得特定的数据。

如：可以通过 ITranslateContext 的方法 getUserdata(String key)取得特定的变量数据。

可以通过配置的参数变量列表按顺序取得变量数据。

如：可以通过 ITranslateContext 的方法 getTranslatorParams()取得变量列表，get

（int）取得变量数据。

3. 单据交换规则定义

本章介绍在“交换规则定义”界面上如何为在信息交换平台中注册的单据进行外系统与NC 系统之间交换规则的定义。信息交换平台除了支持内置单据的标准外系统数据以及 U8 系统数据的集成之外，利用单据交换规则定义扩展对其他外部系统（Sibeil,SAP 等等）的第三方数据的集成的支持。关于单据交换规则的定义，我们分两部分来介绍，基础部分介绍如何完整地为一张单据配置交换规则，高级部分介绍单据交换规则中复杂字段及其对应结构的配置。

我们结合一个主子类型的单据的交换规则的配置，来介绍配置信息交换平台交换规则文件的一般知识。

3.1. 校验文件全局配置

我们建立了一个简单的主子表测试单据，单据类型是“exsystem”。在 NC 系统里为之建立 exsystem 的元数据，如下所示：

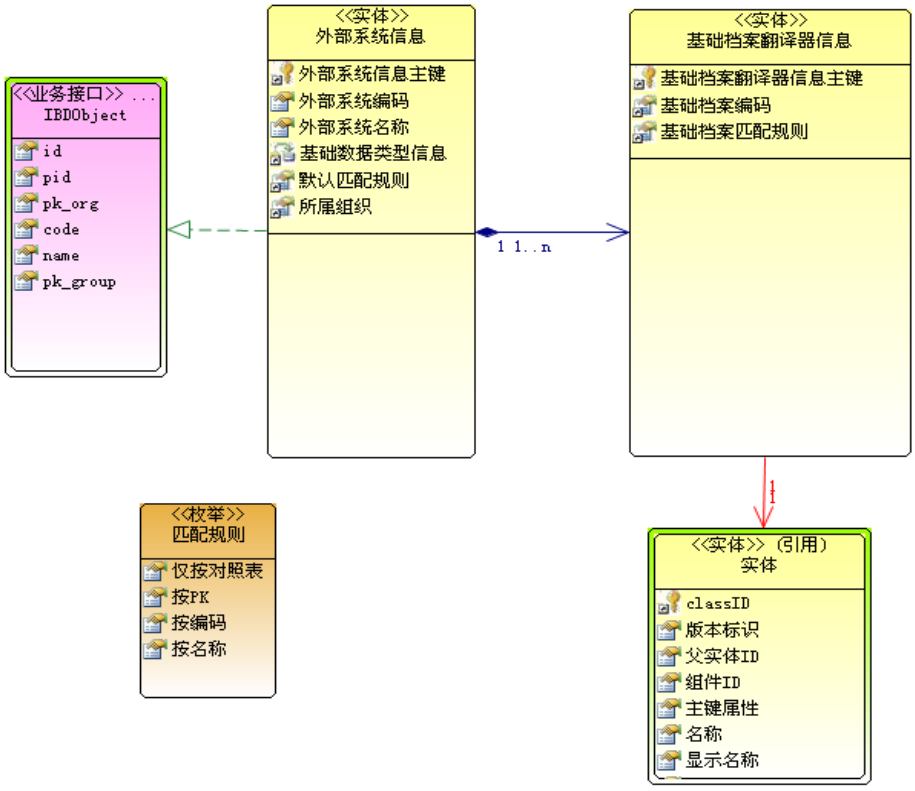


图 3.1.1 主子单据元数据

借助于我们的插件开发向导，可以根据元数据实体生成 XML 的交换规则。图 3.1.2 就是根据该元数据信息自动生成的交换规则。但由于外系统数据与 NC 标准数据之间在名称、结构和语义上的差异，需要通过手工配置来修改自动生成的单据交换规则，以保证内外系统交换时数据的完整性和有效性。

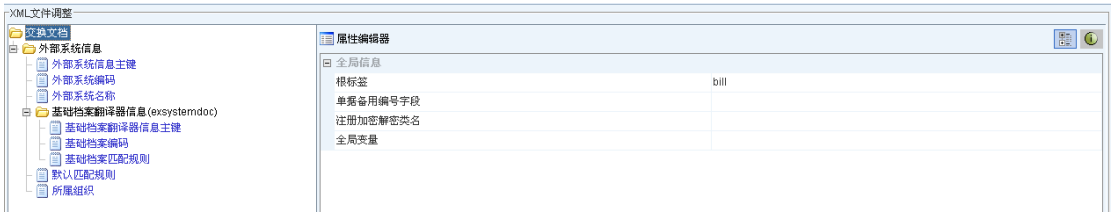


图 3.1.2 可编辑交换规则轮廓

现在假设有一外部的 XML 数据样本文件，如图 1.2 所示。

我们将单据根标签<billdata>定义的元素称为单据数据元素，一个外系统数据文档可以包含多张单据，也就是说可以有多个根标签打头的单据数据元素。由单据表头标签<exsystem_EX>定义的元素称为表头记录元素，一个单据数据元素只包含一个单据表头元素。


```
<?xml version="1.0" encoding='UTF-8'?>
<ufinterface account="develop" billtype="exsystem" filename="example.xml" isexchange="Y" receiver="yk" replace="Y" roottag="billdata" sender="1101">
    <billdata id="">
        <exsystem_EX>
            <pk_exsystem>
            </pk_exsystem>
            <!--外部系统编码,最大长度为40,类型为:String-->
            <exsystemcode>a</exsystemcode>
            <!--外部系统名称,最大长度为500,类型为:String-->
            <exsystemname>a</exsystemname>
            <exsystemdoc>
                <item>
                    <pk_exsystemdoc>
                    </pk_exsystemdoc>
                    <!--基础档案编码,最大长度为36,类型为:String-->
                    <basicdatatypecode>a</basicdatatypecode>
                    <!--基础档案匹配规则,最大长度为1,类型为:Integer-->
                    <basicdoctranslator>0</basicdoctranslator>
                </item>
            </exsystemdoc>
            <!--默认匹配规则,最大长度为1,类型为:Integer-->
            <deftranslator>0</deftranslator>
            <!--所属组织,最大长度为20,类型为:String-->
            <pk_org>a</pk_org>
        </exsystem_EX>
    </billdata>
</ufinterface>
```

图 3.1.3 外部 XML 数据文件样本

由于外系统数据文件的单据根标签为<billdata>，我们首先需要将该交换文档的根标签属性值设置为 billdata。如图 3.1.4 所示。

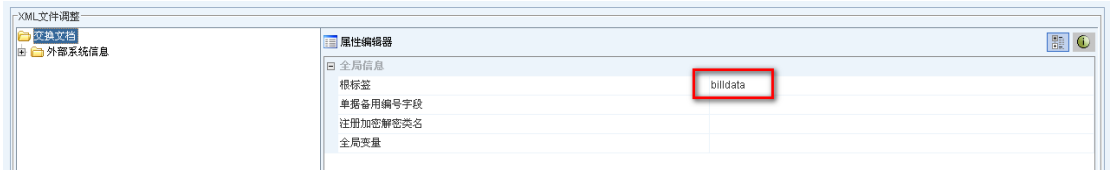


图 3.1.4 修改交换文档根标签

【单据号备选字段】用于在单据没有明确指定单据号时，将该字段的内容作为单据号，插件开发人员可选；

【注册加密解密类名】见第六章。

3.2. 表记录的配置

表头记录定义了 NC 系统里单据表头 VO 的信息及与外系统文件相应单据表头元素的对照关系。如图 3.1.1.1 所示。

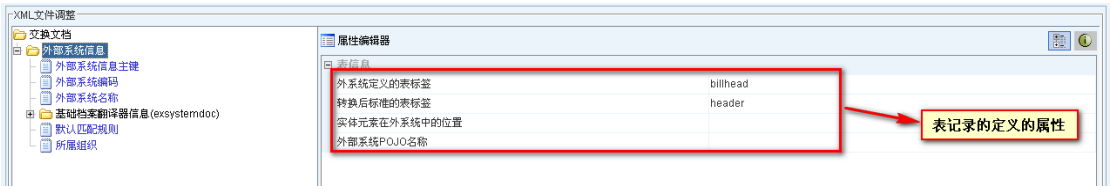


图 3.1.1.1 名为 billhead 的表头记录定义

【外系统定义的表标签】是外系统数据文件中单据表头元素的标签名，在上述数据文件中是<exsystem_EX>，因此需要将该属性值设置为 exsystem_EX。如果数据文件中单据表头元素的标签名与默认生成的【外系统定义的表标签】项属性值不同，必须将该值修改为单据表头元素的标签名，否则就会在导入的过程中报找不到单据表头元素的错误。如图 3.1.1.2 所示。

【转换后标准的表标签】是生成 NC 标准 XML 文件时使用的标签名，由 NC 统一发布，用户一般不应该做修改。

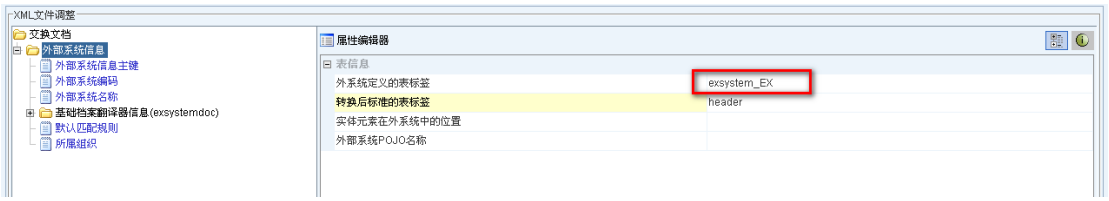


图 3.1.1.2 设置表头记录“外系统定义的表标签”属性项的值

3.3. 字段属性项的配置

接下来，我们需要配置表记录中的每个字段。先介绍两个概念：字段元素和实体元素。DOM 树中的叶子节点（简单元素）我们称之为简单字段元素，因为一般情况下它与我们 NC 系统中的一个数据对象的属性或者数据库表中的一个字段对应。除了简单字段元素之外，还有复杂字段元素。关于复杂字段元素的介绍请参考高级篇。图 3.1.3.1 示例了数据文件中的简单字段元素。

由字段元素（简单字段元素或者复杂字段元素）组成的父元素我们称之为实体元素。

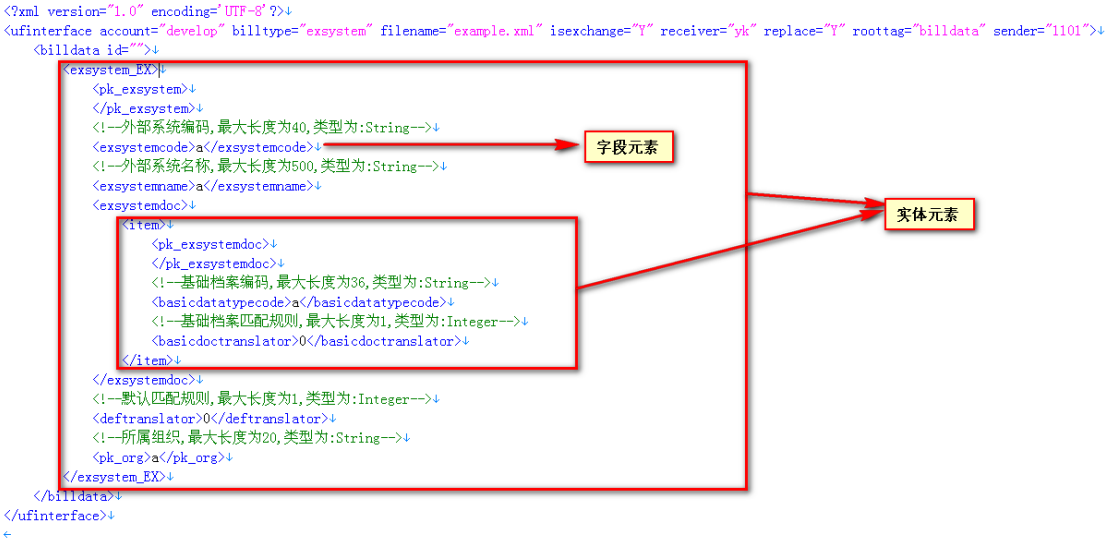


图 3.1.3.1 数据文件中简单字段元素示例

下面介绍字段的各个属性的含义和配置：

【该字段在 NC 的名称】

这个字段在 NC 的数据结构中的名称，这个名称是由 NC 系统的元数据决定的，由校验文件自动生成时自动填充，用户一般不需要修改它。

【该字段在外系统中的名称】

表示字段元素的标签名。数据交换的本质是数据映射，让一个系统中有意义的数据变为另一个系统中同样有意义的数据，但两者的名称可能不一样。在这个地方我们要正确填写外系统数据文件中跟 NC 中某字段对应的字段元素的标签名，否则就会丢失信息。如表体记录定义的字段 exsystemcode 默认的外系统名称为< exsystemcode>，与数据文件中相应字段元素的标签名< excode>不一致，需要将该字段的这个属性值设置为相应字段元素的标签名“excode”，否则在转换过程中就会丢失这个字段的值。

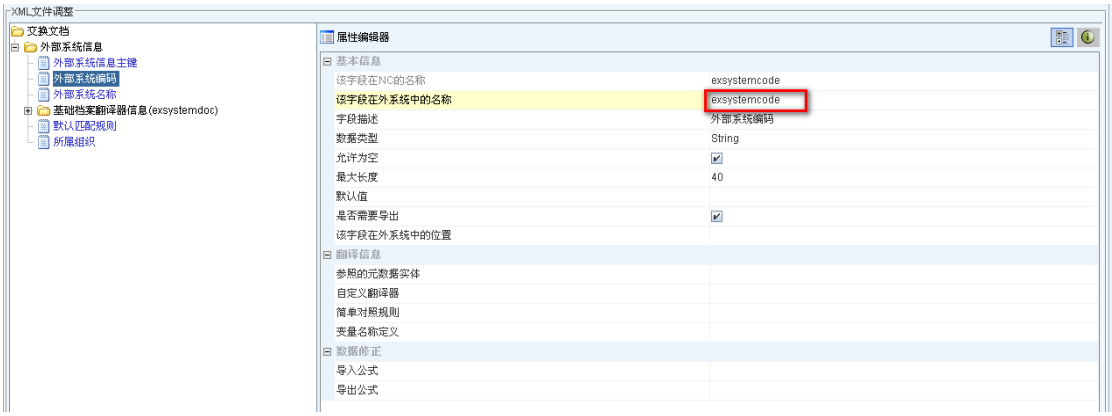


图 3.1.3.3 校验文件默认生成的“该字段在外系统中的名称”

```
<?xml version="1.0" encoding='UTF-8'?>↓
<ufinterface account="develop" billtype="exsystem" filename="example.xml" isexchange="Y" receiver="yk" replace="Y" roottag="billdata" sender="1101">↓
    <billdata id="">↓
        <exsystem_EX>↓
            <pk_exsystem>↓
            </pk_exsystem>↓
            <!--外部系统编码,最大长度为40,类型为:String-->↓
            <excode>a</excode>↓
            <!--外部系统名称,最大长度为500,类型为:String-->↓
            <exsystemname>a</exsystemname>↓
            <exsystemdoc>↓
                <item>↓
                    <pk_exsystemdoc>↓
                    </pk_exsystemdoc>↓
                    <!--基础档案编码,最大长度为36,类型为:String-->↓
                    <basicdatatypecode>a</basicdatatypecode>↓
                    <!--基础档案匹配规则,最大长度为1,类型为:Integer-->↓
                    <basicdoctranslator>0</basicdoctranslator>↓
                </item>↓
            </exsystemdoc>↓
            <!--默认匹配规则,最大长度为1,类型为:Integer-->↓
            <deftranslator>0</deftranslator>↓
            <!--所属组织,最大长度为20,类型为:String-->↓
            <pk_org>a</pk_org>↓
        </exsystem_EX>↓
    </billdata>↓
</ufinterface>↓
```

图 3.1.3.4 数据文件中简单字段的标签名

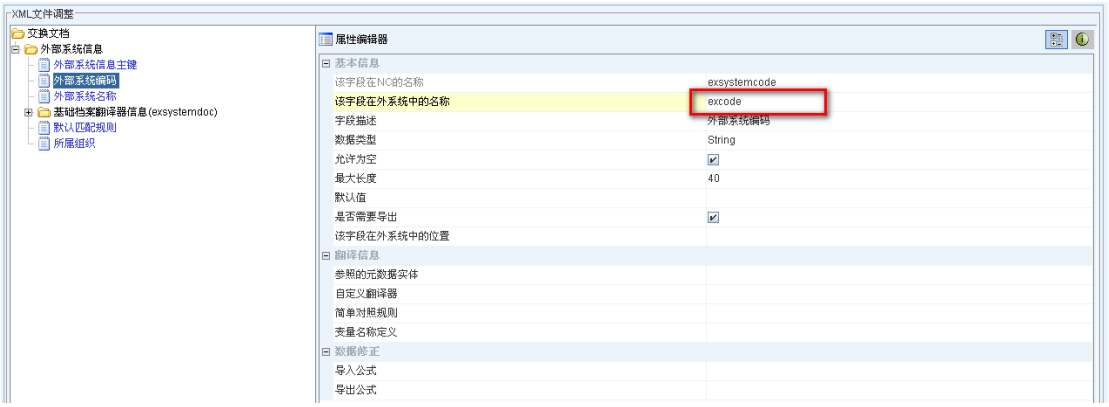


图 3.1.3.5 修改字段的“该字段在外系统中的名称”属性

【字段描述】

对字段的详细描述，使易于理解。校验文件管理界面左侧树上的字段节点的名称显示为字段描述设置的值。

【数据类型】

字段在 NC 的数据结构中的数据类型，在转换过程中用于校验外系统数据。可以设置的值包括字符串、整型、UFBoolean、UFDouble、日期、时间戳等等。通过元数据自动生成。

【允许为空】

字段值是否允许为空，在转换过程中用于校验外系统数据的合法性。通过元数据自动生成。

【最大长度】

字段值的最大长度，在转换过程中用于校验外系统数据的合法性。通过元数据自动生成，一般不用修改。

【默认值】

通过该属性为字段定义一个默认值，当外系统数据文件中没有该字段的值，那么取此默认值导入到 NC 系统中。适合于 NC 内的非空字段，但外系统不具备相应字段元素值的情形，比如导入 U860 的部门档案、会计科目时，U860 相应数据文件均没有公司字段元素，所以需要修改数据文件，或者在校验文件中设置默认值，相对来说，在校验文件中设置默认值显得一劳永逸一些。下面的图例说明了如何设置默认值以及默认值的作用。

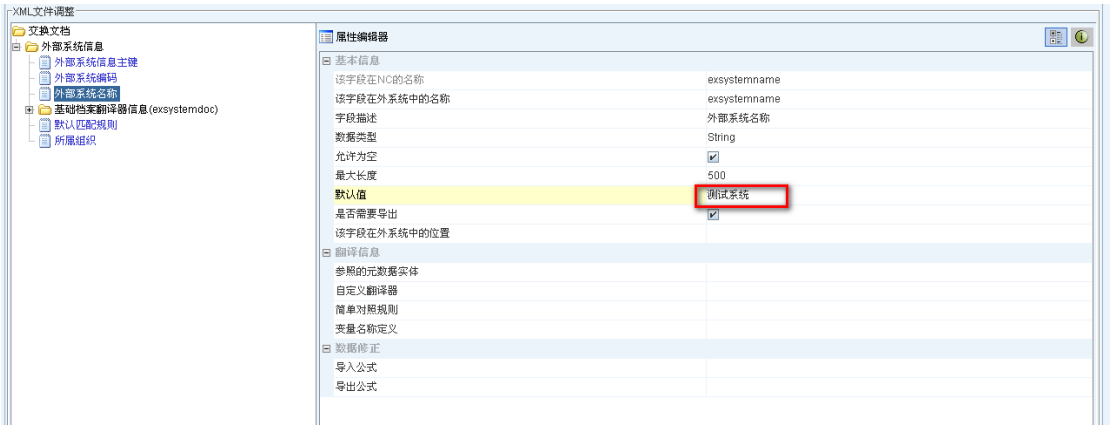


图 3.1.3.12 为字段定义默认值属性

```
<exsystem_EX>↓
    <pk_exsystem>↓
    </pk_exsystem>↓
    <!--外部系统编码,最大长度为40,类型为:String-->↓
    <excode>test01</excode>↓
    <!--外部系统名称,最大长度为500,类型为:String-->↓
    <exsystemname></exsystemname>↓
    <exsystemdoc>↓
        <item>↓
            <pk_exsystemdoc>↓
            </pk_exsystemdoc>↓
            <!--基础档案编码,最大长度为36,类型为:String-->↓
            <basicdatatypecode>a</basicdatatypecode>↓
            <!--基础档案匹配规则,最大长度为1,类型为:Integer-->↓
            <basicdoctranslator>0</basicdoctranslator>↓
        </item>↓
    </exsystemdoc>↓
    <!--默认匹配规则,最大长度为1,类型为:Integer-->↓
    <deftranslator>0</deftranslator>↓
    <!--所属组织,最大长度为20,类型为:String-->↓
    <pk_org>a</pk_org>↓
</exsystem_EX>↓
```

图 3.1.3.13 定义默认值字段的转换前数据

```
<exsystem_EX>↓
    <pk_exsystem>↓
    </pk_exsystem>↓
    <!--外部系统编码,最大长度为40,类型为:String-->↓
    <excode>test01</excode>↓
    <!--外部系统名称,最大长度为500,类型为:String-->↓
    <exsystemname>测试系统</exsystemname>↓
    <exsystemdoc>↓
        <item>↓
            <pk_exsystemdoc>↓
            </pk_exsystemdoc>↓
            <!--基础档案编码,最大长度为36,类型为:String-->↓
            <basicdatatypecode>a</basicdatatypecode>↓
            <!--基础档案匹配规则,最大长度为1,类型为:Integer-->↓
            <basicdoctranslator>0</basicdoctranslator>↓
        </item>↓
    </exsystemdoc>↓
    <!--默认匹配规则,最大长度为1,类型为:Integer-->↓
    <deftranslator>0</deftranslator>↓
    <!--所属组织,最大长度为20,类型为:String-->↓
    <pk_org>a</pk_org>↓
</exsystem_EX>↓
```

图 3.1.3.14 定义默认值字段的转换后数据

【是否需要导出】

数据导出时是否导出该字段

【字段在外系统中的位置】

该项用于指定当前字段元素在整个单据元素中的位置，一般情况下，字段元素都是其父元素即实体元素的子元素，然而当该字段元素没有遵循这个常规，如在某些情况下字段元素是其实体元素的兄弟结点，或者在是其他节点的子节点，这时候就需要设置这个属性。

设置字段位置时可以用相对路径（相对路径的默认起点是当前字段元素的父元素），也可以使用绝对路径（绝对路径必须以_root 开头，_root 表示当前单据元素节点，而不是<ufinterface>元素节点）。路径由外系统元素标签名或者符号“^”构成，之间用记号“|”分隔，标签名表示找子元素，“^”符号表示寻找父元素。如“^|a”表示先找到当前实体元素（即当前字段元素的父元素）的父元素，然后再在其下找名为 a 的元素。

下面我们用例子说明这个属性项。假设现在我们的数据文件如下图所示，简单字段元素<deftranslator>并不是简单实体元素<exsystem_EX>的子元素，而是它的兄弟元素，是单据元素<billdata>的子元素。这种情况下我们就需要设置字段 deftranslator 的【字段在外系统中的位置】项。因为字段 deftranslator 对应的字段元素并不是实体元素（此处也是单据表头元素）的子元素节点，而是其兄弟节点，所以将【字段在外系统中的位置】项值设置为“^”，这样就会从实体元素的父元素（此处也是单据元素）取得该字段元素。

```
<?xml version="1.0" encoding='UTF-8'?>↓
<ufinterface account="develop" billtype="exsystem" filename="example.xml" isexchange="Y" receiver="yk" replace="Y" roottag="billdata" sender="1101">↓
    <billdata id="">↓
        <!--默认匹配规则,最大长度为1,类型为:Integer-->↓
        <deftranslator>0</deftranslator>↓
        <exsystem_EX>↓
            <pk_exsystem>↓
            </pk_exsystem>↓
            <!--外部系统编码,最大长度为40,类型为:String-->↓
            <excode>test01</excode>↓
            <!--外部系统名称,最大长度为500,类型为:String-->↓
            <exsystemname>测试系统</exsystemname>↓
            <exsystemdoc>↓
                <item>↓
                    <pk_exsystemdoc>↓
                    </pk_exsystemdoc>↓
                    <!--基础档案编码,最大长度为36,类型为:String-->↓
                    <basicdatatypecode>a</basicdatatypecode>↓
                    <!--基础档案匹配规则,最大长度为1,类型为:Integer-->↓
                    <basicdoctranslator>0</basicdoctranslator>↓
                </item>↓
            </exsystemdoc>↓
            <!--所属组织,最大长度为20,类型为:String-->↓
            <pk_org>a</pk_org>↓
        </exsystem_EX>↓
    </billdata>↓
</ufinterface>↓
```

图 3.1.3.21 需要定义字段“该字段在外系统中的位置”属性的数据

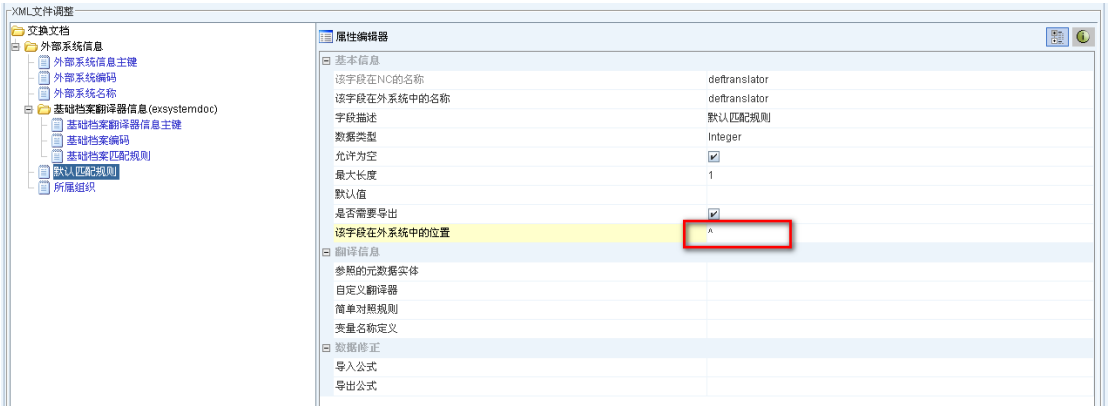


图 3.1.3.22 为字段定义“该字段在外系统中的位置”属性

```
<?xml version="1.0" encoding='UTF-8'?>↓
<ufinterface account="develop" billtype="exsystem" filename="example.xml" isexchange="Y" receiver="yk" replace="Y" roottag="billdata" sender="1101">↓
    <billdata id="">↓
        <exsystem_EX>↓
            <pk_exsystem>↓
            </pk_exsystem>↓
            <!--外部系统编码,最大长度为40,类型为:String-->↓
            <excode>test01</excode>↓
            <!--外部系统名称,最大长度为500,类型为:String-->↓
            <exsystemname>测试系统</exsystemname>↓
            <exsystemdoc>↓
                <item>↓
                    <pk_exsystemdoc>↓
                    </pk_exsystemdoc>↓
                    <!--基础档案编码,最大长度为36,类型为:String-->↓
                    <basicdatatypecode>a</basicdatatypecode>↓
                    <!--基础档案匹配规则,最大长度为1,类型为:Integer-->↓
                    <basicdoctranslator>0</basicdoctranslator>↓
                </item>↓
            </exsystemdoc>↓
            <!--默认匹配规则,最大长度为1,类型为:Integer-->↓
            <deftranslator>0</deftranslator>↓
            <!--所属组织,最大长度为20,类型为:String-->↓
            <pk_org>a</pk_org>↓
        </exsystem_EX>↓
    </billdata>↓
</ufinterface>↓
```

图 3.1.3.23 根据字段的位置属性转换后的数据

【参照的元数据实体】

该元素定义了当前字段属于何种基本档案。因为如果当前字段是基础档案数据，那么和NC 系统交换时需要进行翻译，将外系统值翻译成 NC 系统基础档案的 PK 值。如对于部门，为“部门档案”，人员则为“人员档案”等等，具体编辑时根据参照选择即可。至于如何为外部系统配置基础数据的翻译策略和设置基础数据硬对照，需要参照信息交换平台的使用手册。

通过元数据自动生成，在元数据中表现为 REF 字段。

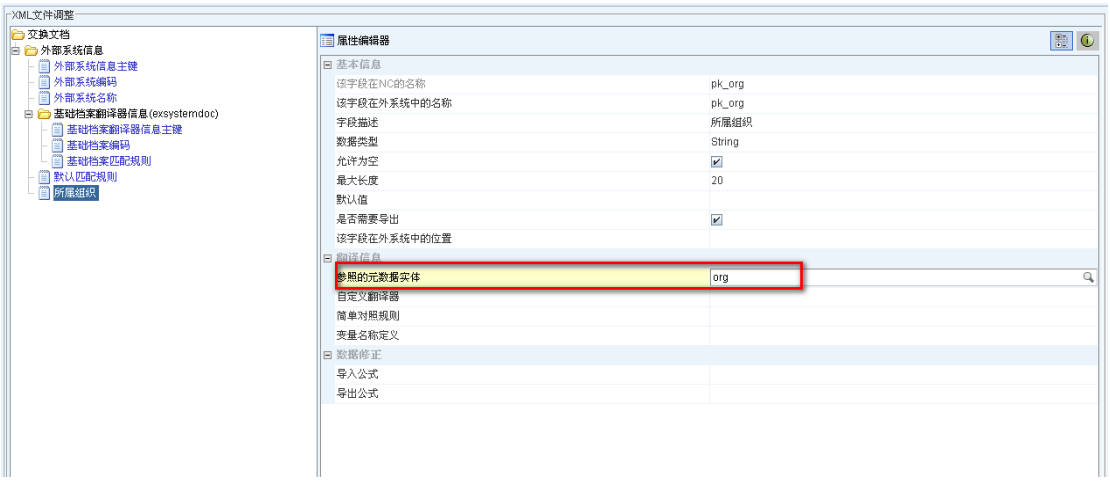


图 3.1.3.9 为字段定义需要参照的基本档案

【自定义翻译器】

当该前字段的翻译不能通过默认的翻译器进行翻译时，需要自定义翻译器进行翻译。只要在“自定义翻译器注册”界面中注册自己的翻译器，并在此处选择对应的翻译器即可。

【自定义翻译器变量参数列表】

自定义翻译器中可能需要特定的变量值用于翻译，如集团，组织等等信息，此处将需要在上下文中使用的变量名按“;”分隔的方式填写，就可以将定义的变量供翻译器使用。此处填写的变量名称，必须是当前字段之前已经翻译的字段，即在左侧树中当前字段上部的字段。

【基础数据对照依赖的组织变量名称】

在进行数据翻译时，需要根据基础数据对照表中的数据进行翻译（参照基础数据对照节点），其中要求根据不同的组织设置不同的对照关系，所以此处需要设定组织变量名称，根据不同的组织进行对照翻译。

将组织字段定义为变量，并将变量名称填写在此处，则在翻译的时候就可以根据不同的组织进行数据对照翻译了。

如果不设置该字段，则使用上下文中的组织进行对照，即 URL 或数据文档头中的 orgcode 设置的接收组织。

【枚举类型】

该元素定义了当前字段属于何种枚举类型。

通过元数据自动生成，在元数据中表现为枚举字段

【变量名称定义】

将当前字段设置为变量，用于自定义翻译器使用，只有设置为变量的数据才能供翻译器的上下文使用。

【简单对照规则】

用于外系统和 NC 系统数据间的简单映射，比如“False=0;True=1”表示如果外系统当前字段的值为 False 那么导入到 NC 系统为 0，如果当前字段的值为 True，那么导入到 NC 系统的值为 1；导出时则恰好相反。下面的图例说明了如何定义字段的规则，将外系统的布尔值对照到 NC 系统的字符串值上。

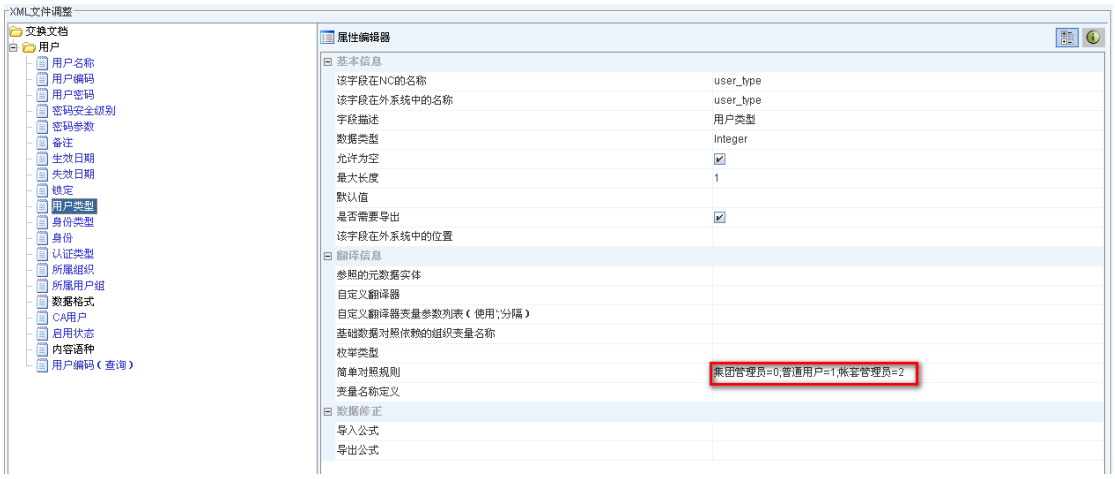


图 3.1.3.6 为字段定义规则属性

下面两张图分别是利用字段规则转换前和转换后的数据：

```
<!--所属组织,最大长度为20,类型为:String-->
<pk_org>rootorg</pk_org>
<!--用户类型,最大长度为1,类型为:Integer-->
<user_type>集团管理员</user_type>
```

图 3.1.3.7 根据规则转换前的数据

```
<!--所属组织,最大长度为20,类型为:String-->
<pk_org>rootorg</pk_org>
<!--用户类型,最大长度为1,类型为:Integer-->
<user_type>0</user_type>
```

图 3.1.3.8 根据规则转换后的数据

【导入公式】

如前所述，在导入数据时，外部系统的 XML 文件首先被转换为符合 NC 标准数据格式的 XML 文件，此 XML 文件紧接着再被转换为 NC 系统内部的数据结构 VO。导入公式主要用于标准 XML 文件到 VO 这一过程中对字段的取值做进一步的处理。下面的图例说明了如何设置字段的导入公式，以及导入公式在单据导入时的作用。

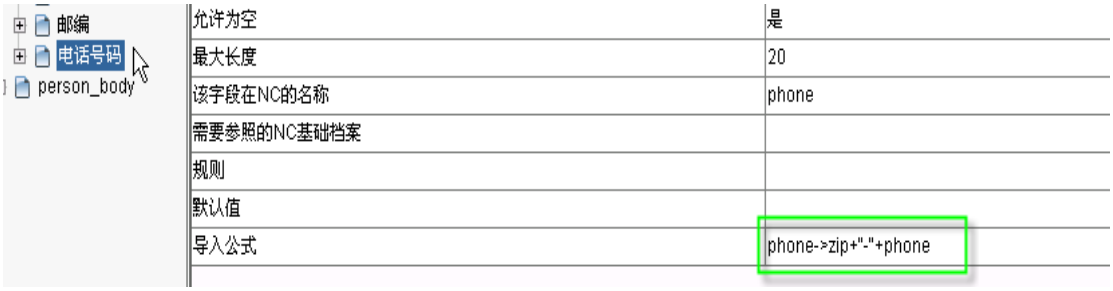


图 3.1.3.15 为字段定义导入公式属性

```
- <customer>
    <street>12345 Happy Lane</street>
    <city>Plunk</city>
    <state>California</state>
    <zip>98059</zip>
    <phone>888.555.1234</phone>
  </customer>
```

图 3.1.3.16 转换前数据

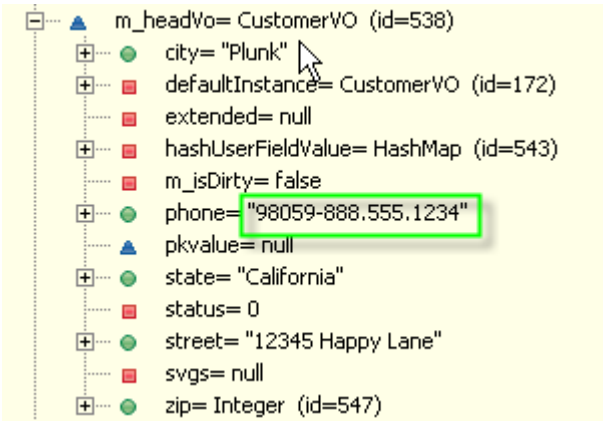


图 3.1.3.17 根据导入公式生成的 VO 内容

【导出公式】

当把 NC 单据(VO 形式)的导出为外部 XML 文件时，会根据该字段的导出公式对 VO 中的数据进行转换，以导出需要的数据。下面的图例说明了如何设置字段的导出公式，以及导出公式在单据导出时的作用。

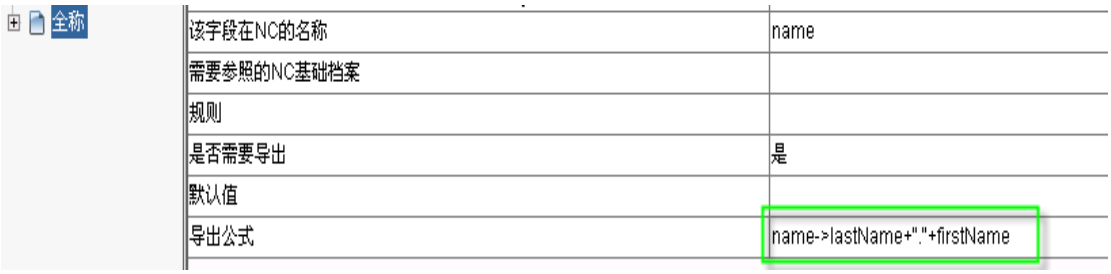


图 3.1.3.18 为字段定义导出公式属性

```
<firstName>Alan</firstName>
<lastName>Turlin</lastName>
```

图 3.1.3.19 导入的数据

```
<first-name>Alan</first-name>
<last-name>Turlin</last-name>
<全称>Turlin.Alan</全称>
```

图 3.1.3.20 导出的数据

4. 基于信息交换平台的单据集成开发

从 V5 版本开始，信息交换平台配套了二次开发向导工具，利用这个工具，可以非常方便的给 NC 系统里还没有集成功能的单据，或者用户自定义的单据，快速添加集成的功能。

从[集成平台]-[数据交换管理]-[集成工具开发]-[插件开发向导]打开信息交换平台的插件开发向导。

下面按照向导的步骤依次介绍开发工具的使用方法及注意事项：

4.1. 注册单据相关信息

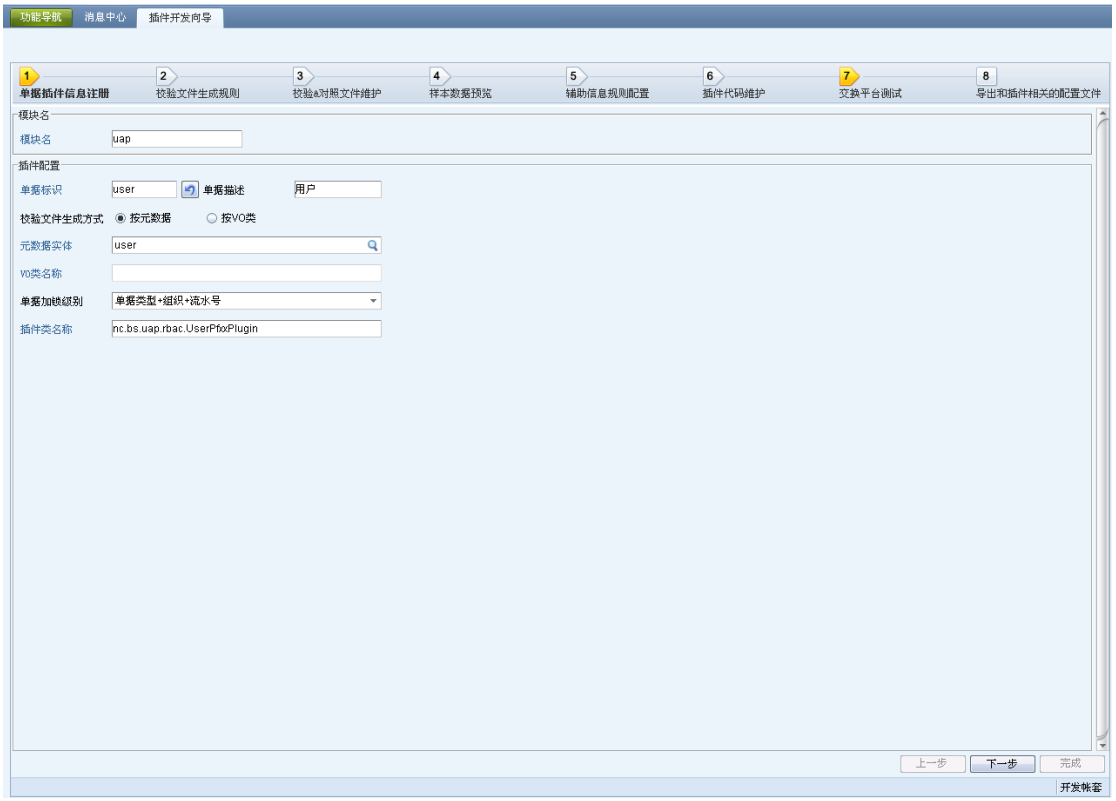


图 4.1.1 单据插件信息注册

说明：

＝＝模块信息＝＝

[模块名] – 单据所属模块，比如部门档案属于”uapbd”模块，凭证属于”gl”模块，模块会影响交换插件最终的部署位置。

＝＝插件配置＝＝

[单据标识] 一般填写相应的单据类型即可，如果没有单据类型，比如基础档案，可以取一个能唯一标识当前档案的名称。比如部门档案取名为”bsdept”

[单据描述] 一般写单据的名称或者简单的功能描述。

[元数据实体] 单据对应在 NC 系统中的元数据实体。

[VO 类名称] VO 类名称，通过输入的 VO 生成交换规则。

注意:必须实现 SuperVO。

[单据加锁级别] 选择框，锁的控制级别，用户控制并发问题，对于大多数单据，如果没有特别的要求，保持默认值即可。关于单据并发控制，详情可以参考 2.6 节。

[插件类名称] 针对该单据，最终的插件处理类名称。这里插件类需要遵循特定的接口，具体可以看后面的[插件类生成及维护]。

4.2. 生成＆配置交换规则定义文件

校验文件配置信息

配置好单据注册相关信息后，就可以自动生成交换规则定义文件的大纲了，之所以叫大纲，是因为生成后的文件，还需要进一步配置，比如对需要翻译的字段配置参照档案等。

首先是设置一些生成参数，以决定是否重新生成，生成通用的规则定义文件，还是针对特定外部系统生成等，如下图所示：

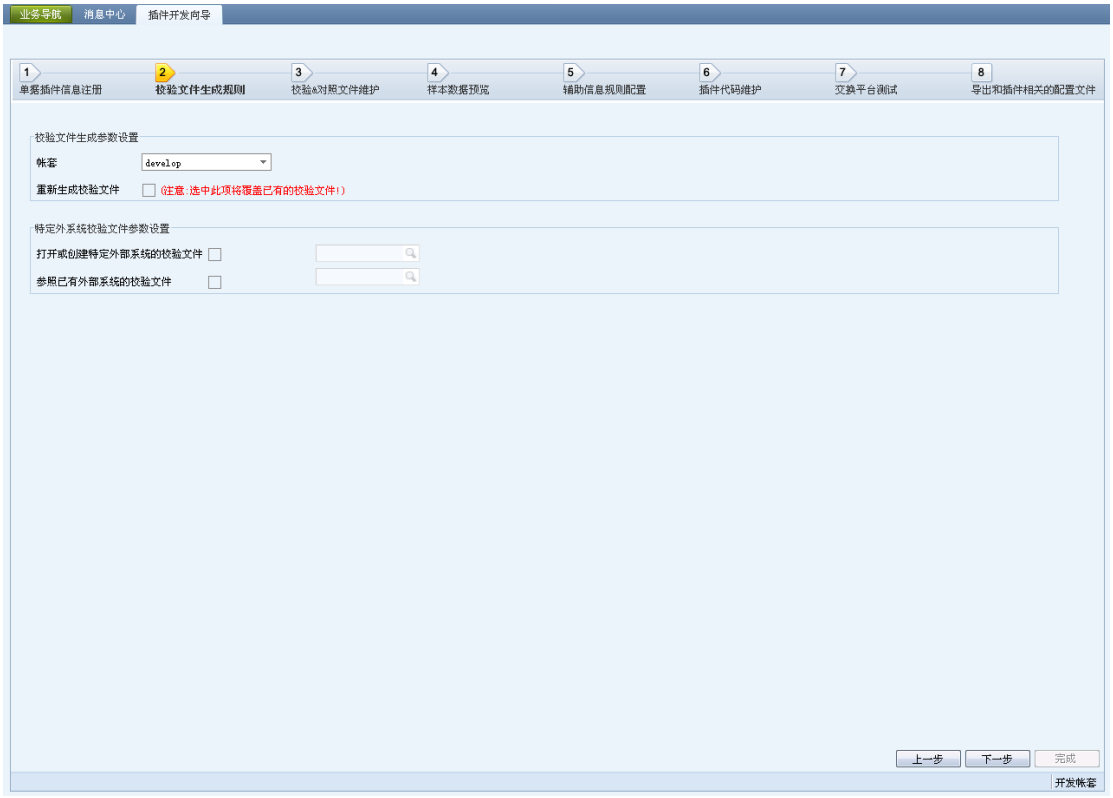


图 4.2.1 校验文件生成规则

生成校验规则

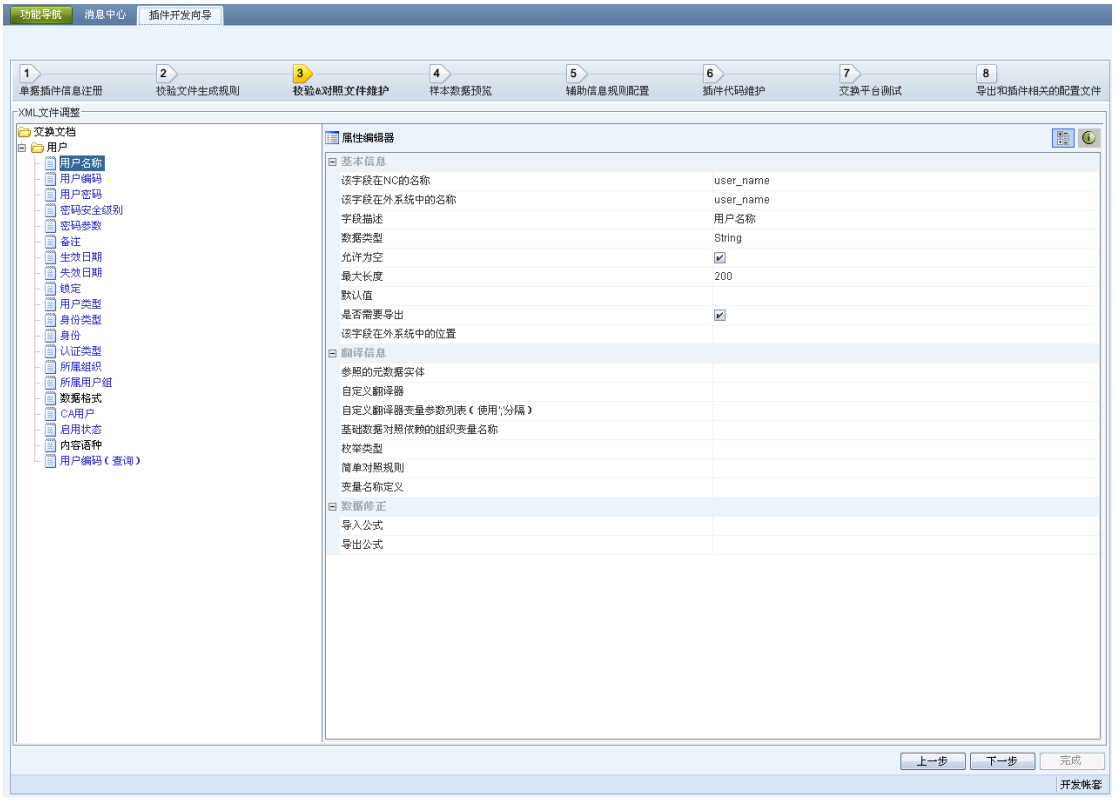


图 4.2.2 校验&对照文件维护

具体如何配置规则定义文件，请参考第三章的内容。

生成样本数据

配置好规则定义文件，就可以生成相应的样本数据，其数据格式符合当前的规则定义，可以直接发送到 NC 系统中。如果修改了规则定义文件，最好重新生成一下样本数据，以免样本数据和规则定义不相符。

根据规则定义生成样本数据，对于外系统没有明确数据格式的情况下，生成的样本数据可以作为外系统生成数据的模板；如果外系统已经有了具体的数据格式，只需要对比一下外系统的数据格式与生成的样本是否一致，便可检验规则定义正确与否。

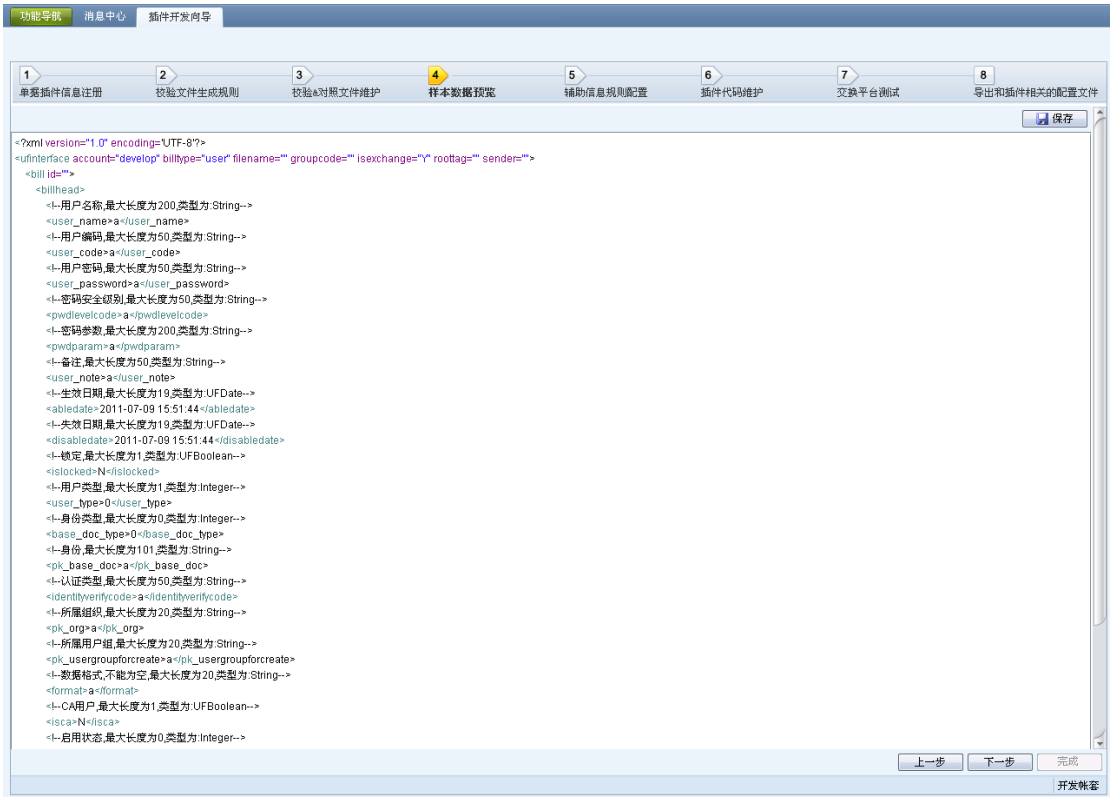


图 4.2.3 样本数据预览

此界面右上方有一保存按钮，可以保存生成的样本数据到本地硬盘。

4.3. 辅助信息项设置

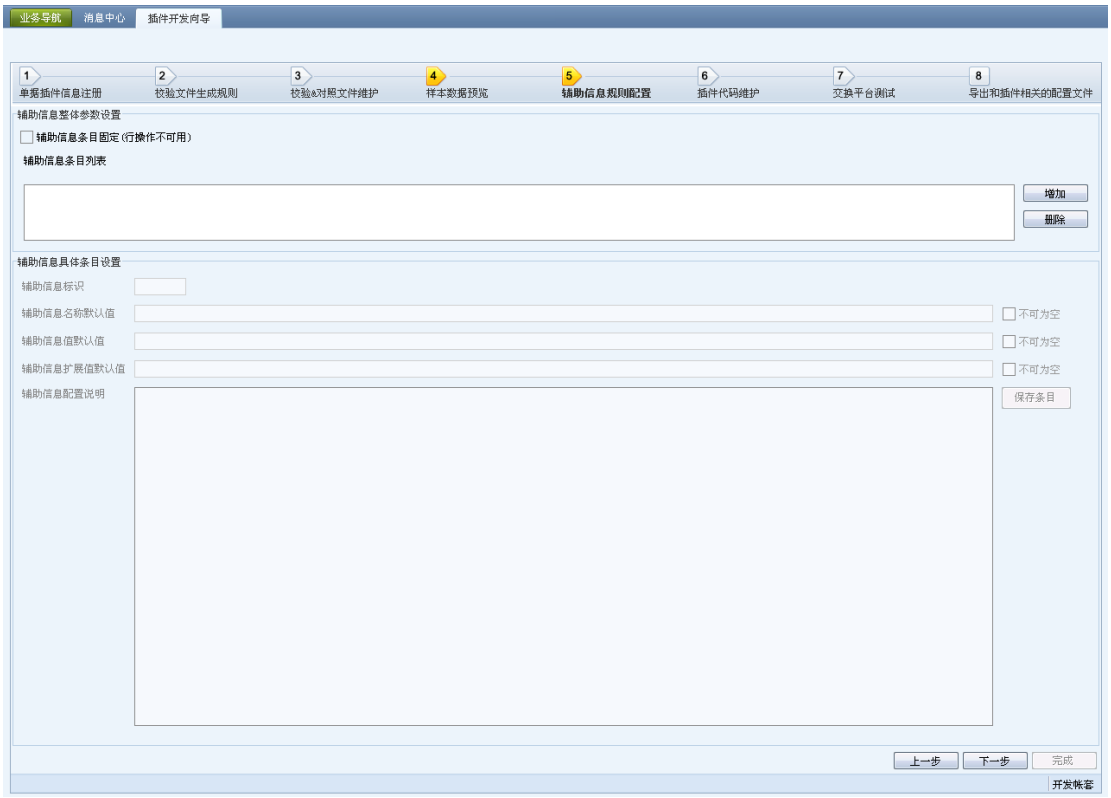


图 4.3.1 辅助信息项设置

利用交换平台做数据交互时，业务数据一般都放在具体的 XML 文档中，但是，发送过程中往往还有一些数据，并不需要也不能写到每个具体的 XML 文档里，这些数据，用户往往需要根据当前的发送方、接受方、单据类型及具体的业务场景进行设定，比如说发送凭证时，针对凭证表头表体的控制参数是根据不同的发送方、接受方、及凭证类别由用户灵活控制的；另外如存货基础档案发送到集团时，用户也需要配置，档案可以分配到那些公司。诸如此类的信息，在交换平台中称之为辅助信息。

辅助信息分设计态和运行态，设计态是指从程序开发的角度，交换平台提供了一个类似数据池的数据存贮区，插件开发人员先定义当前单据所需要的辅助信息项目，比如上面所说的凭证控制信息、存货的分配公司等，插件开发人员定义好这些项目之后，在[集成平台]-[数据交换管理]-[信息交换平台]-[辅助信息设置]界面，用户就可以在运行态时配置这些项目的值，最终在业务插件类里，用户配置好的这些值被交给插件使用。简单的说就是插件开发人员借助信息交换平台辅助信息这个数据池，存放一些特定的项目，在运行态时收集用户的设

置，最终在插件处理这些用户的设置信息。

4.4. 插件代码维护

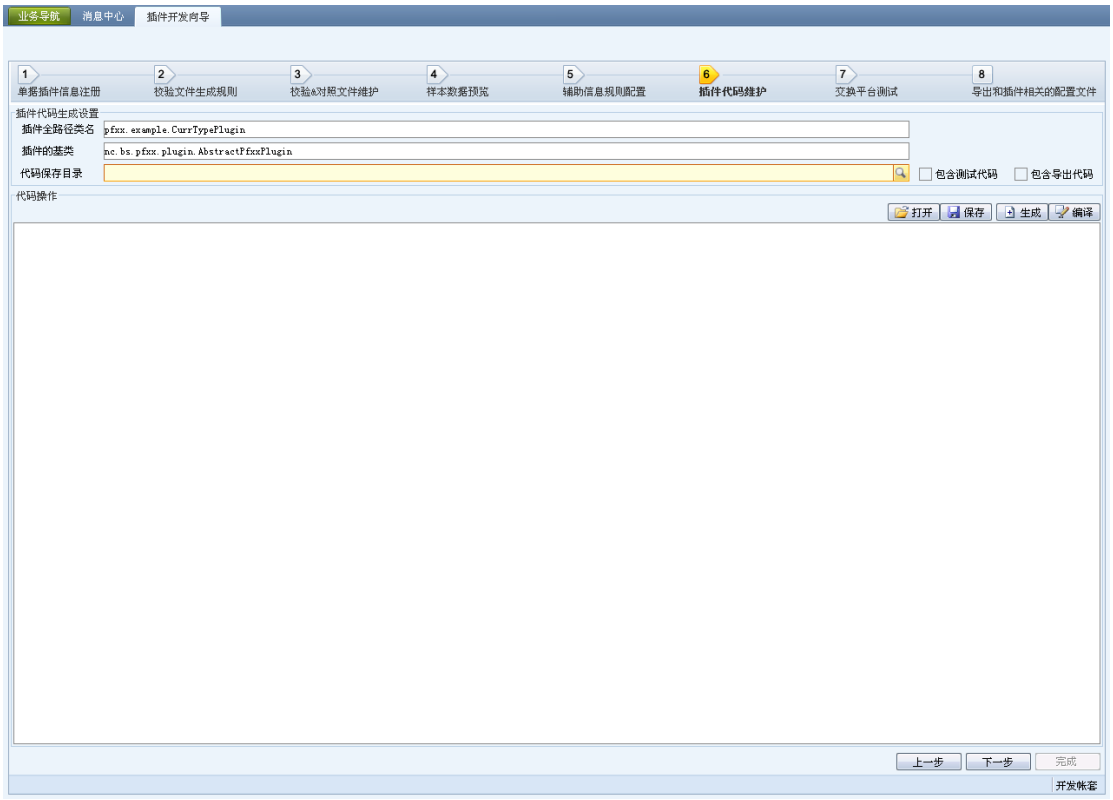


图 4.4.1 插件代码维护

【插件全路径类名】根据单据注册信息自动带过来，不可以再修改。插件基类为 AbstractPfxxPlugin，一般也无需修改。

【包含测试代码】如果选中此项，生成的代码中会包含测试代码，主要是将转换后的 VO 序列化到后台，最终在前台进行展现，以便开发人员观察转换后的 VO 是否正确，便于定位错误。仅适合在测试时使用，最终代码开发时一定要删除测试代码。

【包含导出代码】如果选中此项，会生成如何将 VO 转换为 XML 文档的代码实例，以便开发人员参考。

选择生成代码的存放路径，便可以按“生成”，生成代码。

这个代码是插件的通用框架，如果有辅助信息或者其他的业务处理，请自行拷贝到 ECLIPSE 修改。

修改好的代码也可以拷贝到这里，进行编译(中间件需要设置参数 enableHotDeployed=true),然后便可以测试了。

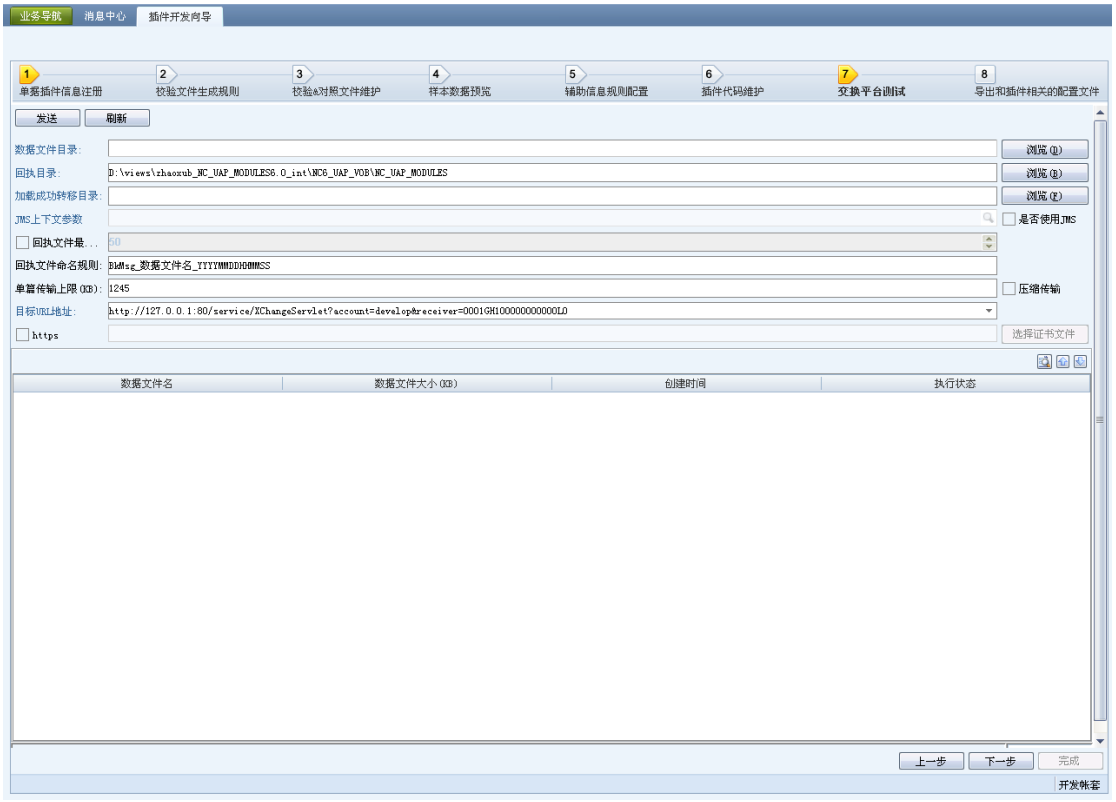
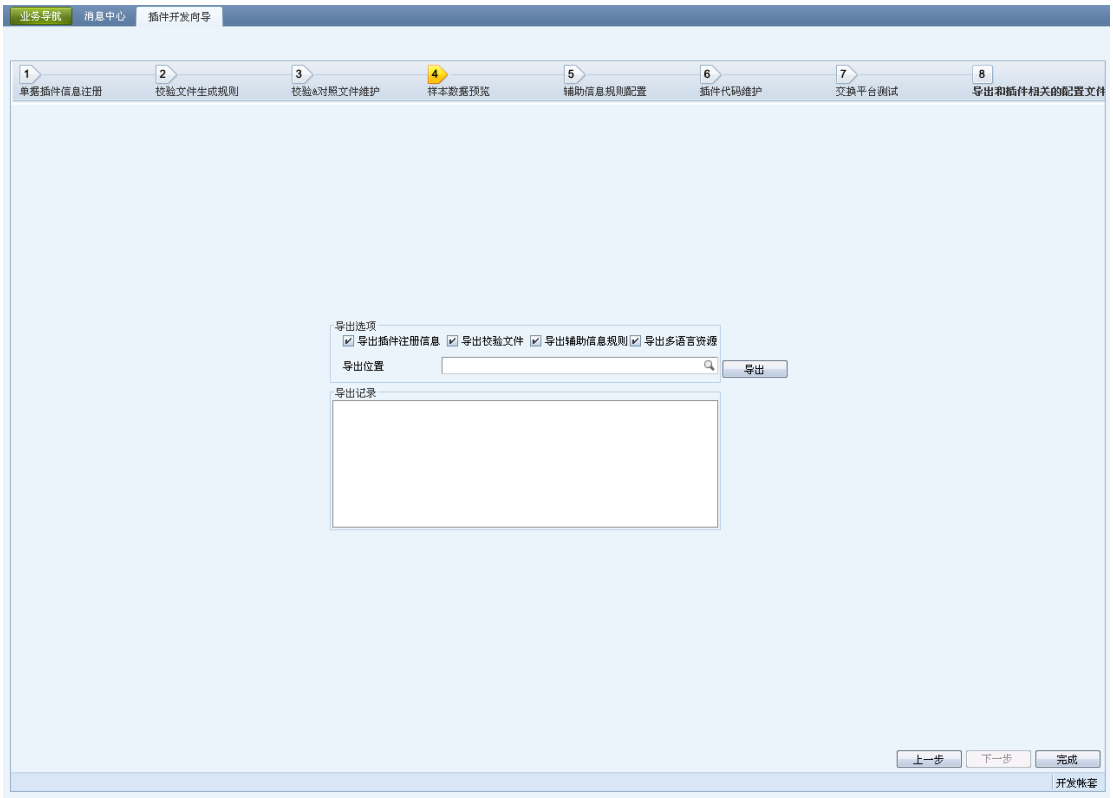


图 4.4.2 交换平台测试

测试 OK 后，可以导出相关的配置文件，提交到 CC 服务器或者其他的测试环境上。



5. 单据集成示例

利用插件开发向导为自定义单据“自定义档案”开发集成功能。

5.1. 问题描述

需要将 K 系统某 XML 格式的自定义档案的数据集成至 NC 系统中来，但是目前 NC 系统没有集成自定义档案 defdoc 的功能，需要定制开发该功能。并且由于 K 系统的自定义档案格式与 NC 系统自定义档案的标准格式不一致，需要修改单据转换规则以使得 K 系统数据能正确导入。

5.2. 设计

在进行具体开发前须先期完成的任务：在信息交换平台注册为 K 系统注册编码为 1101 的外部系统，准备好 K 系统的自定义档案样本数据 defdoc.xml 文件（具体请参见附录 2，样本数据格式由外系统决定）。然后你需要执行以下步骤：

单据插件信息注册

单据转换规则定义

插件代码编写和部署

修改单据转换规则

将数据加载至 NC 系统

5.3. 具体开发指导

5.3.1. 单据插件信息注册

以组织（或者集团）身份登陆 NC 系统。

选择集成平台>数据交换管理>集成工具开发>插件开发向导。

在单据插件信息注册界面，在模块名输入框中输入 uapeai。

在单据标识输入框中输入 defdoc，在单据描述输入框中输入自定义档案，在单据加锁级

别下拉框中选择单据类型+组织+流水号，在插件类名称输入框中输入nc.bs.pfxx.plugins.DefdocPlugin。在元数据实体选择对应的元数据。

点击下一步。

5.3.2. 单据转换规则定义

9．在校验文件生成规则界面上，选择重新生成校验文件多选框。

10．点击下一步。

11．在校验&对照文件维护界面上，对自动生成的单据转换规则进行定义，主要工作：首先剔出每个记录中不需要的字段，然后修改每个字段的“最大长度”、“允许为空”、“需要参照的元数据实体”三个属性值。不需要字段主要有记录本身的主键字段（如表头记录的pk_defdoc、primaryKey 等），对字段属性的修改需要参照该单据的单据模版（如表头记录的字段 pk_org 参照组织目录）。

表头记录的 pk_defdoclist 字段节点，选择参照的元数据实体 defdoclist;

选择 pk_org 节点，在修改属性需要与接收方一致的值为是。

12．点击下一步，保存单据转换规则（校验文件）。

13．点击下一步，跳过样本数据预览界面。

14．点击下一步，跳过辅助信息规则配置界面。

5.3.3. 插件代码编写和部署

15．在插件代码维护界面上，选择代码导出路径，不必选择包含导出代码多选按钮。

16．点击生成按钮，开始编辑插件类 DefdocPlugin 的代码。

17．在方法 processBill(Object, ISwapContext, AggxsysregisterVO)中，编写如下代码：

```
DefdocVO defdocvo = (DefdocVO) vo;
IDefdocService imp = NCLocator.getInstance().lookup(IDefdocService.class);
defdocvo.setDatatype(new Integer(1));
imp.insertDefdocs(swapContext.getOrgPk(), new DefdocVO[] { defdocvo });
return null;
```

18．点击编译按钮，将.java 文件编译成.class 文件。

19. 点击下一步。

20. 点击下一步，跳过交换平台测试界面。

21. 点击完成，跳过导出和插件相关的配置文件界面，退出集成开发向导。

5.3.4. 修改单据转换规则

22. 在 NC 主界面上，选择集成平台>数据交换管理>校验文件管理。

23. 在交换规则定义界面上，选择打开菜单。

24. 在打开对话框上，在单据类型参照输入框中参照值 defdoc。

25. 在打开对话框上，点击确定按钮。

26. 在左树右表的交换规则定义界面上，点击左树的交换文档节点，修改右表根标签值为 defdoc。

27. 在左树上，点击表头节点，修改右表中外系统定义的表标签值为 defdoc。

28. 在左树上，双击表头节点，打开表头文件夹。

29. 在左树上，点击 pk_org 节点，修改右表中该字段在外系统中的名称为 org。

30. 在左树上，点击 pk_defdoclist 节点，修改右表中该字段在外系统中的名称为 defdoc。

33. 在菜单栏，选择另存为菜单。

34. 在另存为对话框上，在外部系统参照输入框中参照值 1101。

35. 在另存为对话框上，点击确定按钮。

5.3.5. 手动加载测试

36. 在 NC 主界面上，选择集成平台>数据交换管理>手动加载界面。

37. 在手动加载界面的数据文件目录中浏览输入 defdoc.xml 文件所在的目录。

38. 在数据文件列表中选中 defdoc.xml 一行。

39. 在菜单栏上，选择发送菜单。

40. 查看发送回执文件。点击“回执查看”按钮，查看回执信息。如果熟悉回执文件的结构，也可以直接打开回执文件查看，如果 resultcode 等于 1 的话，表示数据加载成功，否则，根据错误提示，修改单据转换规则、数据文档头属性值以及基础数据对照等配置信息。

41. 如果数据加载成功，在 NC 主界面上，选择开发平台〉开发配置工具〉自定义档案

定义，查看数据是否插入成功。

6. 安全

本章介绍如何为在信息交换平台中进行单据内容的加解密

6.1. 加密类编写

加密接口。

在 502 信息交换平台里增加了加密接口，内容如下：

```
package nc.bs.pfxx.plugin;


import nc.vo.pfxx.exception.PfxxException;


/**
 * 交换系统加密接口
 * @author Larrylau
 */
public interface ICryptProcessor{
    /**
     * 将 BASE64 串转换为密文，然后解密
     * @author Larrylau
     * @param base64text
     * @return structured xml document snippet
     * @throws PfxxException
     */
    public String decrypt(String base64text) throws PfxxException;


    /**
     * 将明文加密后，必须将密文转换为 BASE64 串
     * @author Larrylau
```

```
     * @param text
     * @return base64text
     * @throws PfxxException
     */
    public String encrypt(String text) throws PfxxException;
}
```

对于加密后的单据内容，一般情况下需要将密文转换成 BASE64 文本编码。同样解密单据内容，先将 BASE64 文本转换成密文然后再解密。

默认实现类

```
package nc.bs.pfxx.plugin;

import java.io.IOException;

import nc.bs.logging.Logger;
import nc.vo.pfxx.exception.PfxxException;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class DefaultCryptProcessor implements ICryptProcessor{
    protected byte[] deBASE64String(String base64text) throws PfxxException{
        try{
            return new BASE64Decoder().decodeBuffer(base64text);
        }catch (IOException e){
            Logger.error("不能解析 base64 文本\r\n" + e.getMessage(), e);
            throw new PfxxException(e);
        }
    }

    protected String getBASE64String(byte[] bytes) throws PfxxException{
```

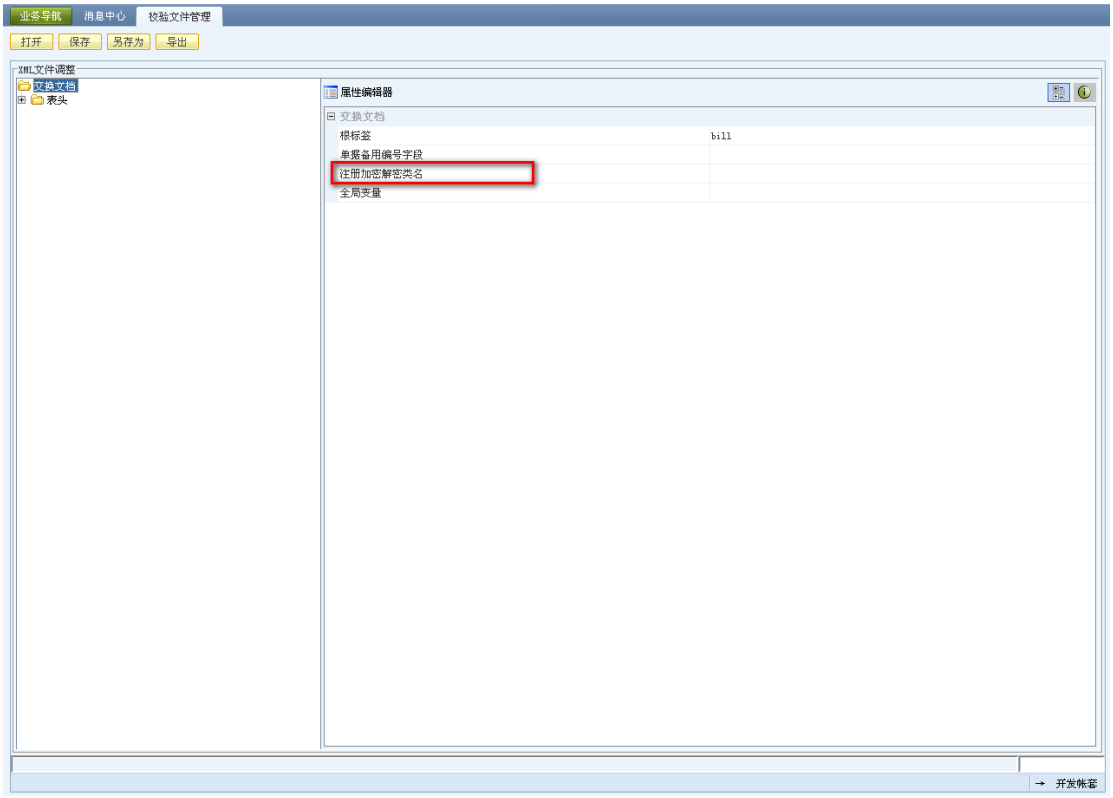
```
            return new BASE64Encoder().encode(bytes);
        }

        /**
         * 将 BASE64 串转换为密文，然后解密
         * @author Larrylau
         * @see nc.bs.pfxx.plugin.ICryptProcessor#decrypt(java.lang.String)
         */
        public String decrypt(String base64text) throws PfxxException{
            return    new String(deBASE64String(base64text));
        }

        /**
         * 将明文加密后，必须将密文转换为 BASE64 串
         * @author Larrylau
         * @see nc.bs.pfxx.plugin.ICryptProcessor#encrypt(java.lang.String)
         */
        public String encrypt(String text) throws PfxxException{
            return getBASE64String(text.getBytes());
        }
}
```

该默认实现不涉及任何加解密内容，用户可以扩展这个默认实现来写自己的加解密类。

6.2. 加密类注册

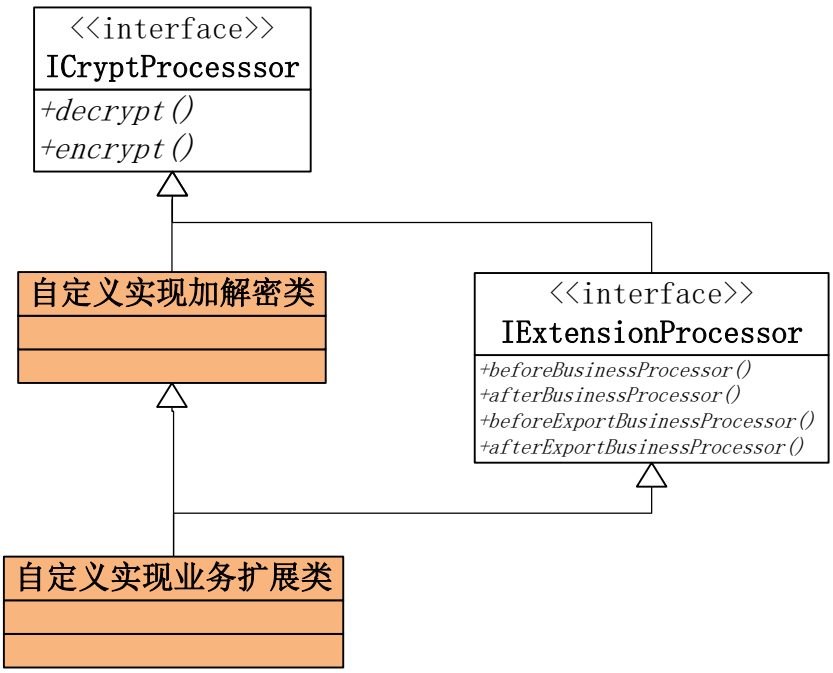
在校验文件管理界面中，打开一个规则定义。在右边的文本框中，填入自己的加密实现类，并把加密实现类放到

${nchome}\modules\uapeai\META-INF\classes 下


业务导航
消息中心
校验文件管理
打开
保存
另存为
导出
XML文件调整
交换文档
表头
属性编辑器
交换文档
根标签
bill
单据备用编号字段
注册加密解密类名
全局变量
开发帐套

7. 扩展

信息交换平台从V55版开始增加了一个扩展接口，提供标准业务插件处理前和处理后，以及单据导出前和导出后的事件处理，二次开发人员可以通过实现此接口扩展内置业务插件没有提供的功能，或者在导出数据时增加业务逻辑，此扩展类由于继承了 ICryptProcessor(参考加密类编写)，所以实现类也建议继承原有的加密实现类，然后修改原来注册的加密类为新的实现类。



新的扩展类具体接口及方法如下：

```
public interface IExtensionProcessor extends ICryptProcessor {


    /**
     * @author Larrylau
     * @param vo
     *                  返回结果 转换后的 vo 数据，在 NC 系统中可能为 ValueObject,SuperVO,AggregatedValueObject,IExAggVO 等。
     * @param aggvo
```

```
     *                辅助信息
     * @param doc
     *                转换后 xml 文档
     * @param context
     *                翻译上下文
     * @throws PfxxException
     */
    public void beforeBusinessProcessor(Object vo, final AggxsysregisterVO aggvo,
Document doc, ISwapContext context)
            throws PfxxException;


    /**
     * @author Larrylau
     * @param resobj
     *                业务插件返回结果
     * @param aggvo
     *                辅助信息
     * @param doc
     *                转换后 xml 文档
     * @param context
     *                翻译上下文
     * @return  返回  resobj
     * @throws PfxxException
     */
    public Object afterBusinessProcessor(Object resobj, final AggxsysregisterVO aggvo,
Document doc,
            ISwapContext context) throws PfxxException;


    /**
```

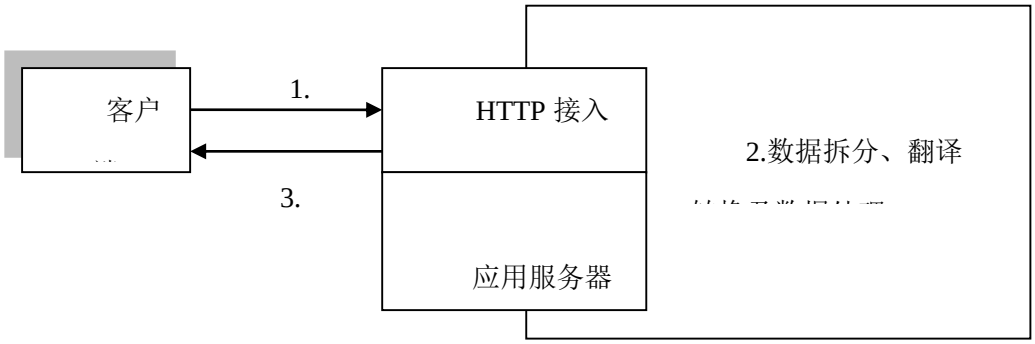
```
        * @author Larrylau
        * @param vo
        *                  需要导出的 vos
        * @param context
        *                  导出上下文
        * @throws PfxxException
        */
       public void beforeExportBusinessProcessor(Object[] vos, ISwapContext context) throws
PfxxException;


       /**
        * @author Larrylau
        * @param doc
        *                  导出的 xml
        * @param context
        *                  导出上下文
        * @return  处理后的  xml
        * @throws PfxxException
        */
       public Document afterExportBusinessProcessor(Document doc, ISwapContext context)
throws PfxxException;
    }
```

8. JMS 及大文件传输模式

8.1. 信息交换平台异步通信解决方案

8.1.1. 信息交换平台现状及存在的问题

信息交换平台目前主要是通过 HTTP/HTTPS 来接受来自各种客户端 post 的数据，接受到数据后，读取 post 请求的数据流，解析为 xml，通过翻译转换后派发给注册的业务处理插件，处理完毕后将结果以 httpResponse 的方式返回给客户端，客户端读取返回结果，判断请求是否成功的被 NC 服务器处理。其处理过程如下图所示：



从以上描述可以看出存在下面的问题：

无法完成负载均衡。

异步请求难以实现。 而对于同步请求，如果出现网络异常或者在做耗时长的数据处理时，客户会陷入漫长的等待中，这将严重影响客户的检验效果。

有些业务本身就是异步的，如定时数据汇总等，这种情况下会让客户感觉很奇怪。

在出现异常时（如网络中断，等待结果超时或者客户端系统重新启动等），不能保证数据一致性。

很难充分地利用服务端的资源——所有客户端来的请求接收完成后必须马上处理，没有机会根据应用服务器上的负载情况进行实时的调度。

8.1.2. 需求分析

从上面的分析可以看出我们需要一个跨集群的通信平台，通过合理的部署，该平台可以：

跨广域网的数据交换能力

高可靠性：对用户每次传输的数据做“唯一标识”，一旦用户数据进入平台，必须提供严格的数据一致性，无论任何类型的异常情况出现客户的数据都不应该重复或者丢失。

高可用性：只要本地部署的节点中有一个可用就可以继续缓存或者向远端传输用户的数据。

负载均衡能力：可以做一定程度的缓冲或者分发以充分利用服务端资源。

异步通信能力

提供一定程度的数据组合逻辑支持

易于扩展

根据以上的分析，消息中间件是最好的选择。本方案也将使用 Active MQ 构建集群通信台来解决外部交换平台所面临的问题。

8.1.3. JMS 简介

(1) 消息系统类型

通常有两种消息类型

发布/订阅（publish/subscribe）

发布/订阅消息系统支持一个事件驱动模型，消息产生者和使用者都参与消息的传递。产生者发布事件，而使用者订阅感兴趣的事件，并使用事件。产生者将消息和一个特定的主题(Topic)连在一起，消息系统根据使用者注册的兴趣，将消息传给使用者。

点对点（Peer to peer）

在点对点的消息系统中，消息分发给一个单独的使用者。它维持一个"进入"消息队列。消息应用程序发送消息到一个特定的队列，而客户端从一个队列中得到消息。

(2) JMS（Java 消息服务）是一个消息交换标准，它答应使用 J2EE 应用程序组件建立、发送、接收和读取消息。它假设分布式通讯拥有自由（free）的连接、是可靠的（reliable）和异步的（asynchronous）。

JMS 应用程序的主要部分是：

- 产生连接的部分和目的地
- 连接
- 对话
- 产生消息的部分
- 使用消息的部分
- 消息

产生连接的部分（ConnectionFactory）是负责建立 JMS 连接的对象。每个 ConnectionFactory 都是 QueueConnectionFactory 或 TopicConnectionFactory 的一个副本（copy）。在点对点的模型中，它使用了 javax.jms.QueueConnectionFactory；在发表－预订模型中，它使用的是 javax.jms.TopicConnectionFactory。

目的地（Destination）——它是队列或主题，这依靠于我们使用了下面哪种模型：javax.jms.Queue 或 javax.jms.Topic。

连接（Connection）——它可能是客户端和服务应用之间的开放的 TCP/IP。它可以被用于建立一个或少量的对话。在你的应用程序能够接收消息前，你必须调用 start()方法。为了暂停发送消息，你需要调用 stop()。

对话（Session）——在 JMS 连接的帮助下建立的对象，被客户端用作发送和接收消息。

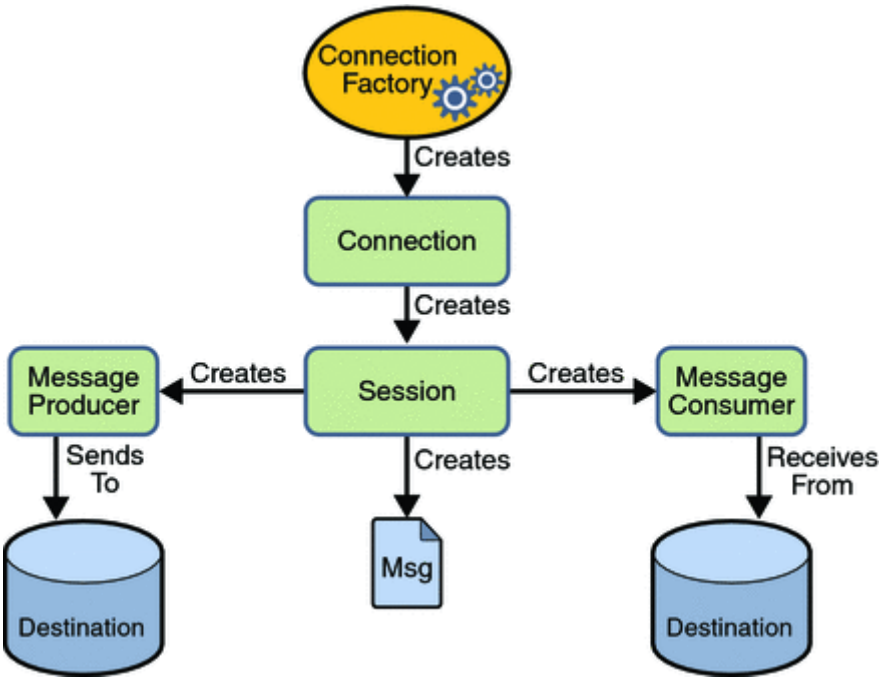
产生消息的部分（MessageProducer）——对话建立的对象，被用于在目的地中发送消息。

使用消息的部分（MessageConsumer）——对话建立的对象，用于接收消息。为了同步接收消息，需要使用 receive()方法。对于异步的情形，使用 MessageListener 和唯一的方法——onMessage()。在该方法中，在定义的消息到达后应该执行一定的操作。

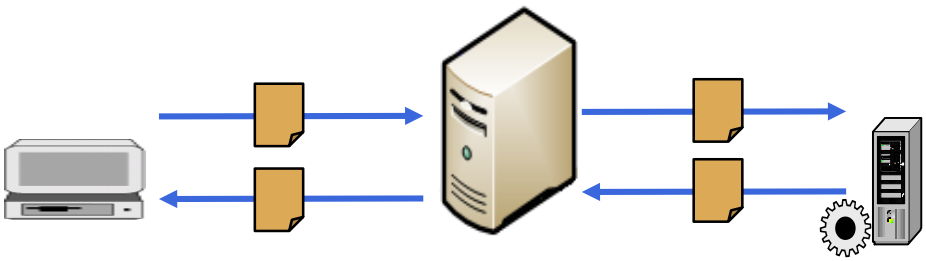
消息（Message）——消息本身。JMS 消息由三个部分组成：

- 消息头
- 属性（不是必要的）
- 消息体（不是必要的）

具体的编程模型如下图



8.1.4. JMS Client 消息交互图



M MQ

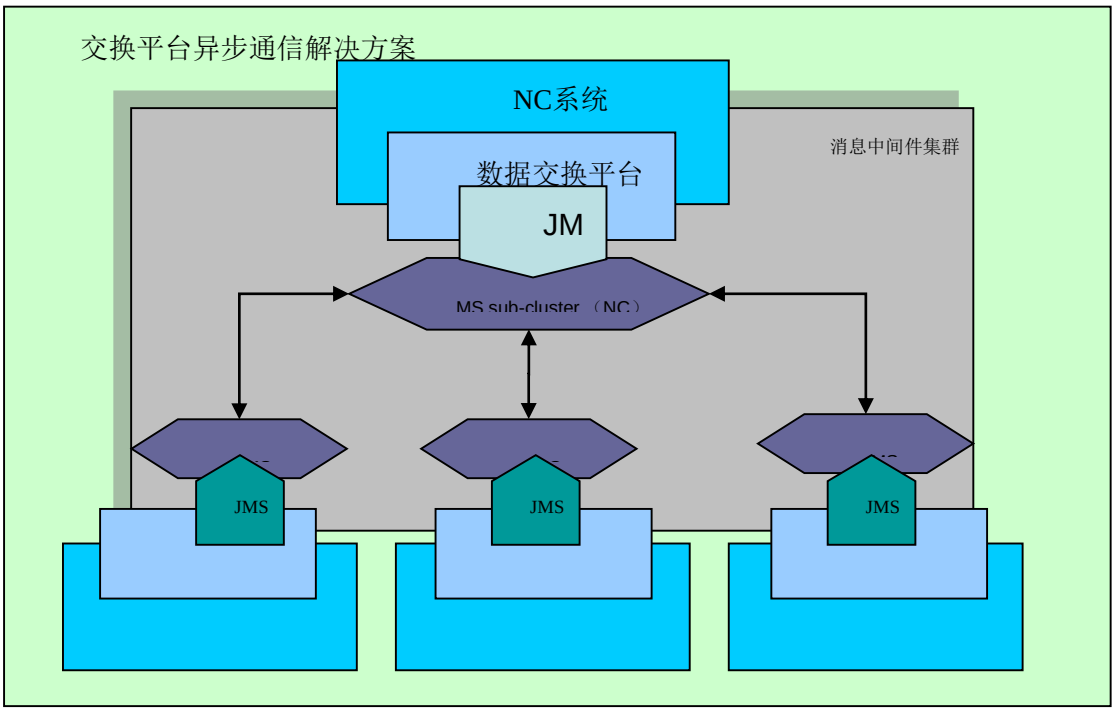
JMS NC

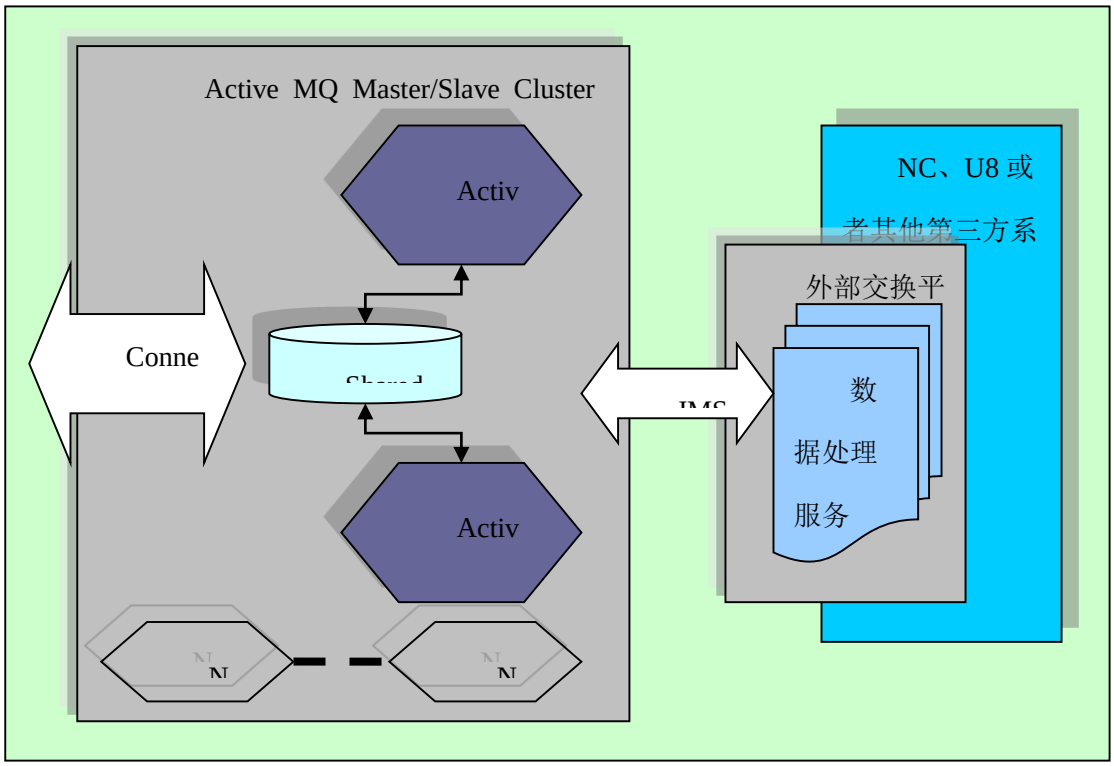
用户利用 JMS Client 框架发送消息（可以是文件，字符串或者一个文档对象等）到 ActiveMQ 服务器，在 NCServer 的交换平台模块中，开发人员增加了对 ActiveMQ Server 的监听功能，从而可以迅速的取得用户发送的信息，把这些消息发送到 NCServer 的相应模块和应用中进行处理，然后返回回执，放入 ActiceMQ Server 的对应队列中，用户可以利用 JMS Client 框架很方便的取得这些回执，然后根据需要进行下一步的处理。

8.1.5. 解决方案

8.1.5.1. 系统架构

整个的系统架构如下图所示：





信息交换平台异步通信解决方案

在本方案中，每一个数据交换平台涉及到的点上都会部署一个或者多个 Active MQ 的节点，然后由这些节点构成一个本地的小集群（子集群），这些节点将共享队列数据，从外部看来就是一个节点（在任何一个时间点上都只有一个节点可用）。同时通过配置网络连接，各个子集群再组成一个更大范围的集群。用户只要按照 JMS 规范编写自己的应用组件（无论客户端还是服务端），就可以把用户的请求（构造成 JMS 消息）发送到本地子集群，这是第一级缓存。一旦进入本地子集群，Active MQ 将保证其可靠，高效的发送到目的地（目的地的子集群就是第二级缓存）。

8.1.5.2. 子集群详细架构：

如上图所示，在一个子集群内，需要部署至少两个以上的 Active MQ 节点，这些节点使用相同的共享存储来保证本地系统的高可用性——只要有一个 Active MQ 节点能够正常工作，那本地的通信平台就可以正常工作。

注：共享存储可以有多个选择：

考虑到性能和稳定性的权衡建议使用共享的文件系统如 NFS。

对可靠性要求较高但性能相对要求不高的情况建议选择使用数据库作为　共享存储。

如果对数据可靠性有最高的要求建议使用第三方的共享存储解决方案。

8.2. JMS 传输模式

如果需要了解 UFMQ 的配置方法，请参照《UFMQ 使用指南》。

下面列出 NC 端的简单配置说明（重启中间生效）：

建 ActvieMQ 服务器

检查并修改 NCHOME\ierp\bin\ncjca-conf.xml

将 j2cResourceAdapter 标签对应的 active 属性修改为“true”

```
<J2C_RAs>↓
  <j2cResourceAdapter jndiName="ActiveMQ.ra" name="ActiveMQ.ra" description="ActiveMQ resource adapter" active="true">↓
    <propertySet>↓
```

将对应的 ServerUrl 修改为对应 MQ 服务的 URL，如图

```
<resourceProperty name="redeliveryUseExponentialBackOff" type="java.lang.Boolean" value="false" description="redeliveryUseExponentialBackOff" requ
<resourceProperty name="ServerUrl" type="java.lang.String" value="tcp://20.10.1.223:61616" description="The URL to the ActiveMQ server that you wa
<resourceProperty name="topicPrefetch" type="java.lang.Integer" value="32766" description="topicPrefetch" required="false"/>↓
```

(3)检查并修改 NCHOME\ierp\bin\ nc-mdb-bind.xml

将 jndiName="PfxxBillMessage"的节点属性 active 修改为“true”。

根据具体要求配置最大线程数。

```
<property name="maxThreads" type="Integer" value="1"/>↓
```

（6）手工加载界面

选中　是否使用 JMS

输入正确的目标 URL 地址　形如：tcp://localhost:61616

添入 JMS 上下文参数

必输项，及其默认值

属性名	说明	默认值	必输项
account	接收帐套	当前帐套	
billType	单据类型		

pk_group	接收集团	当前集团	Y
receiver	接收方	当前组织	
sender	发送方		Y
jmsUrl	Jms 方式传输的 url		
split	全局分割名称		
sendQ	JMS 发送队列		Y
receiveQ	JMS 接收队列(客户端相关，全局唯一)		Y
dualMessageSendQ	Jms 发送小事务模式		
dualCount	Jms 发送小事务模式每次提交单据的个数		

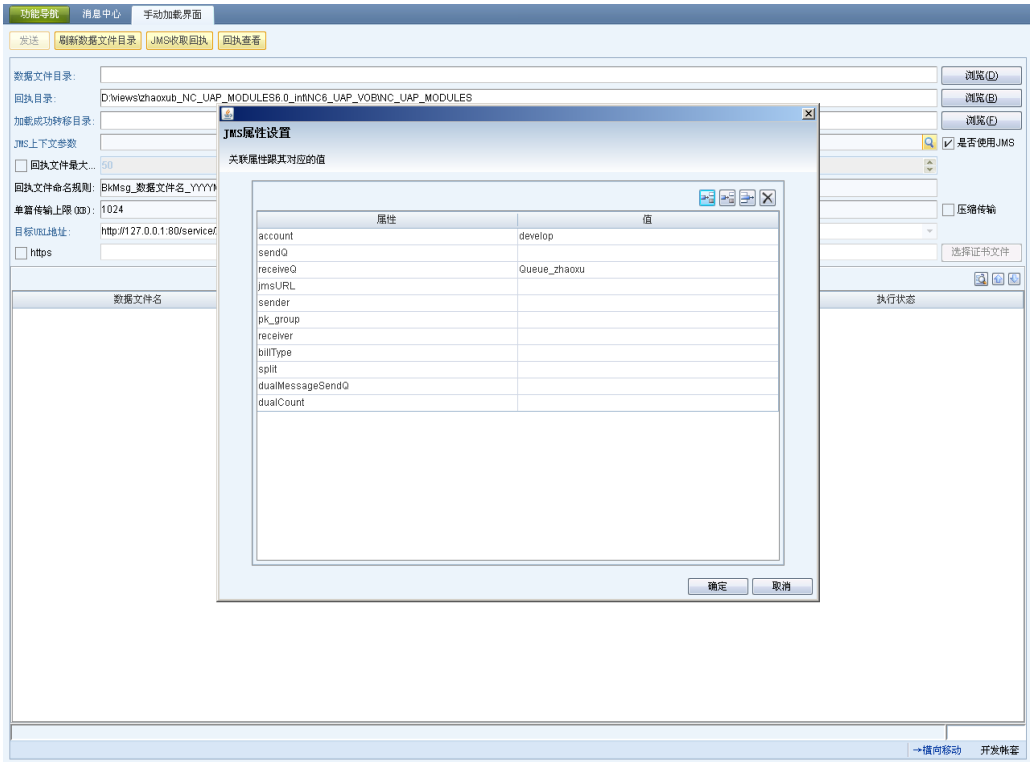
Split 全局分割字符串，对当前所有发送文件有效。为空，取待发送 XML 文件 ufinterface 标签中 roottag 属性值，不能为空

待发送文件，只能发送一次

等待，接收 JMS 回执。

JMS Client UI

为了方便用户进行消息的发送和接收，本组件还提供了图形界面供用户使用。



8.3. JMS Client For NC6.x

JMS Client for NC6.x(以下简称 JMS Client)是为了满足 NC5.6 及以后版本信息交换平台的内在需求，对 JMS 进行了封装，向用户提供了非常友好易用的 API，从而使用户能够方便的发送，接收，处理消息。

在 NC5.6 之前，NC Server 的 web 交换平台只能接受 http 的请求，这样就会带来两个局限，第一个局限是 http 是点对点协议，它不提供对多个接收方的并发请求能力；第二个局限是 http 本身只支持同步通讯，用户发送消息以后程序必须一直等待请求响应，占用宝贵的通讯资源直到接收到响应信息。而引入 JMS Client 以后，新的交换平台增加了对异步消息的支持（JMS Client 本身仍然支持同步模式），用户发送消息以后，程序无需等待应答，用户可以订阅相应的主题信息，当应答信息到达服务器后，服务器会主动根据订阅的主题来寻找对应的用户，通知用户响应信息已经到达，从而极大的提高的系统的效率。引入 JMS Client 以后，交换平台既能支持同步操作也能支持异步操作，既能进行点对点方式，又能进行发布/订阅方式，更多的消息传送方式使得用户在与 NC 之间的通讯选择最合适的方式时，提供了更灵活的选择。

通过 JMS Client 发送的 JMS 消息还可以设置优先级，使得对时间敏感的信息流有更多的控制，还可以作为定义服务通讯相关质量的方式。管理员通常使用这种功能调整服务级别，完善整体服务性能。这样在网络拥堵、系统临时宕机或者灾难恢复时，能确保那些拥有相关优先级的关键系统能够正常运行并进行通讯。当应用程序或者网络出现间歇性的失效甚至停止运行，JMS 能确保消息从发送者(生产者)传送到接收者(消费者)，因此，JMS Client 的引入给 N C 带来的好处是显而易见的，对用户来说，不但多了一种通讯的选择，而且非常友好，极大提高了用户的效率。

8.3.1. JMS Client API

JMS Client 主要是封装了 JMS 的相关 API，向用户提供利用 JMS 规范跟 NC 服务器交互的统一接口，方便操作，简单易用。

getMessageFileID

```
public static java.lang.String getMessageFileID(javax.jms.Message  message)
```

throws javax.jms.JMSException

获取消息文件 iD

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

getMessageReplyQueueName

public static java.lang.String getMessageReplyQueueName(javax.jms.Message message)

throws javax.jms.JMSException

获取应答队列名

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

getMessageID

public static java.lang.String getMessageID(javax.jms.Message message)

throws javax.jms.JMSException

获取消息 ID

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

getBatchID

public static java.lang.String getBatchID(javax.jms.Message message)
throws javax.jms.JMSException

获取消息批次 ID

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

getMessageSeqID

public static java.lang.String getMessageSeqID(javax.jms.Message message)
throws javax.jms.JMSException

发送消息序号

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

getMessageSendClientName

public static java.lang.String getMessageSendClientName(javax.jms.Message message)
throws javax.jms.JMSException

获取客户端发送 ID

参数：

message -

返回：

抛出：

javax.jms.JMSException

---

createConnectionByReflect

```
public static javax.jms.Connection createConnectionByReflect(java.lang.String url)
                                                       throws javax.jms.JMSException
```

根据指定 url 创建 ActvieMQ 连接，并启动

参数：

url -

返回：

抛出：

javax.naming.NamingException

javax.jms.JMSException

---

getFilePathMD5

```
public static java.lang.String getFilePathMD5(java.lang.String filePath)
```

根据文件全路径计算 MD5

参数：

filePath -

返回：

---

sendFileDual

```
public static int sendFileDual(java.lang.String url,
                               java.lang.String filePath,
                               java.lang.String queueName,
                               java.lang.String replyQueueName,
                               java.lang.String dualQueueName,
                               int dualLength,
```

```
ava.util.Map<java.lang.String,java.lang.String>  requestMap)
                          throws java.lang.Exception
```

已过时。

参数：

url -

filePath -

queueName -

replyQueueName -

dualQueueName -

dualLength -

requestMap -

返回：

抛出：

java.lang.Exception

---

sendFile

```
public static int sendFile(java.lang.String  url,
                           java.lang.String  filePath,
                           java.lang.String  fileID,
                           java.lang.String  batchID,
                           java.lang.String  clientID,
                           java.lang.String  queueName,
java.lang.String  receiptQueueName,
java.util.Map<java.lang.String,java.lang.String>  requestMap,
                           java.lang.String  dualQueueName,
                           int  dualLength,
                          boolean  single)
                           throws java.lang.Exception
```

抛出：

java.lang.Exception

---

sendFile

```
public static int sendFile(javax.jms.Connection  conn,
                    java.lang.String  filePath,
                    java.lang.String  fileID,
                    java.lang.String  batchID,
                    java.lang.String  clientID,
                    java.lang.String  queueName,
                    java.lang.String  receiptQueueName,
                    
java.util.Map<java.lang.String,java.lang.String>  requestMap,
                    java.lang.String  dualQueueName,
                    int  dualLength,
                    boolean  single)
                    throws java.lang.Exception
```

抛出：

java.lang.Exception

---

sendFile

```
public static int sendFile(java.lang.String  url,
                    java.lang.String  filePath,
                    java.lang.String  queueName,
                    java.lang.String  replyQueueName,
                    
java.util.Map<java.lang.String,java.lang.String>  requestMap)
                    throws java.lang.Exception
```

已过时。

发送一个文件

参数：

url -

filePath -

queueName -

replyQueueName - 里的参数信息，将覆盖 xml 中 ufinterface 标签的值

requestMap -

返回：

抛出：

java.lang.Exception

---

createMessageID

public static java.lang.String createMessageID()

创建唯一消息 ID

返回：

---

createTSID

public static java.lang.String createTSID()

---

sendReceiptMessage

```
public static void sendReceiptMessage(javax.jms.Session  session,
                                      java.lang.String  queueName,
                                      java.lang.String  messageText,
                                      java.lang.String  fileID,
                                      java.lang.String  seqID,
                                      java.lang.String  clientID,
                                      boolean  isSuccessed,
                                      java.lang.String  batchID,
```

java.util.Map<java.lang.String,java.lang.String> map)

throws javax.jms.JMSException

发送回执消息 其中 batchID+fileID+clientID 能确定同一批次发送的消息 s

参数:

session -

queueName -

messageText -

fileID -

batchID -

seqID -

clientID -

isSuccessed -

map -

抛出:

javax.jms.JMSException

---

createEAIReceiptMessage

public static <T extends javax.jms.Message> T createEAIReceiptMessage(T message,

boolean isSuccessed,

java.lang.String fileID,

java.lang.String batchID,

java.lang.String seqID,

java.lang.String clientID,

java.util.Map<java.lang.String,java.lang.String> map)

throws javax.jms.JMSException

创建回执消息 其中 batchID+fileID+clientID 能确定同一批次发送的回执消息 s

类型参数:

T -

参数：

message -

isSuccessed -

fileID -

batchID -

seqID -

clientID -

map -

返回：

抛出：

javax.jms.JMSException

---

sendTextmessage

```
public static void sendTextmessage(javax.jms.Session  session,
                                   javax.jms.MessageProducer  producer,
                                   java.lang.String  replyQueueName,
                                   java.lang.String  messageText,
                                   java.lang.String  messageID,
                                   java.lang.String  fileID,
                                   java.lang.String  batchID,
                                   java.lang.String  seqID,
                                   java.lang.String  clientID,
                                   java.util.Map<java.lang.String,java.lang.String>  map)
                            throws javax.jms.JMSException
```

发送消息

参数：

messageText - 消息正文

session - Jms Session

producer - Jms Producer

messageID - 消息 ID

seqID - 序列 ID

fileID - 消息文件 ID

clientID - 客户端 ID

map - 消息属性

抛出：

javax.jms.JMSException

---

sendTextmessage

```
public static void sendTextmessage(javax.jms.TextMessage  txtMsg,
                                   javax.jms.MessageProducer  producer,
                                   java.lang.String  replyQueueName,
                                   java.lang.String  messageText,
                                   java.lang.String  messageID,
                                   java.lang.String  fileID,
                                   java.lang.String  batchID,
                                   java.lang.String  seqID,
                                   java.lang.String  clientID,
                                   java.util.Map<java.lang.String,java.lang.String>  map)
                            throws javax.jms.JMSException
```

发送 jms 消息

参数：

txtMsg -

producer -

replyQueueName -

messageText -

messageID -

fileID -

batchID -

seqID -

clientID -

map -

抛出：

javax.jms.JMSException

---

createEAIMessage

```
public static <T extends javax.jms.Message> T createEAIMessage(T  message,
                                        java.lang.String  replyQueueName,
                                        java.lang.String  messageID,
                                        java.lang.String  fileID,
                                        java.lang.String  batchID,
                                        java.lang.String  seqID,
                                        java.lang.String  clientID,
                                        java.util.Map<java.lang.String,java.lang.String>  map)
                                 throws javax.jms.JMSException
```

创建一个携带 UAP-EAI 平台特定属性的 JMS 消息。其中 batchID+fileID+clientID 能确定同一批次发送的消息 s

参数：

message -

replyQueueName - 回执返回的 Queue 名称

messageID - 消息 ID（唯一）

fileID - 消息所属文件 ID

batchID - 消息所属批次 ID

seqID - 消息所属批次序列 ID

clientID - 发送者 ID

map - 其他附加属性

返回：

txtMessage

抛出：

javax.jms.JMSException

---

sendDualTextmessage

```
public static void sendDualTextmessage(javax.jms.Session  session,
                                       javax.jms.MessageProducer  producer,
                                       java.lang.String  fileID,
                                       java.lang.String  clientID,
                                       java.lang.String  batchID,
                                       int  seqCount)
                                throws javax.jms.JMSException
```

抛出：

javax.jms.JMSException

---

receive

```
public static void receive(javax.jms.Session  session,
                           java.lang.String  fileID,
                           java.lang.String  batchID,
                           java.lang.String  clientID,
                           java.lang.String  queueName,
                           javax.jms.MessageListener  listener)
                    throws javax.jms.JMSException
```

参数：

session - 由外部程序关闭

fileID -

clientID -

queueName -

listener -

抛出：

javax.jms.JMSException

---

receive

public static java.util.List<javax.jms.TextMessage> receive(java.lang.String url,

java.lang.String batchID,

java.lang.String fileID,

java.lang.String clientID,

java.lang.String queueName,

long timeout)

throws javax.naming.NamingException,

javax.jms.JMSException

接收回执

参数：

url -

filePath -

queueName -

timeout -

返回：

抛出：

javax.naming.NamingException

javax.jms.JMSException

---

receive

```
public static java.util.List<javax.jms.TextMessage>
receive(javax.jms.Connection conn,
                                    java.lang.String batchID,
                                    java.lang.String fileID,
                                    java.lang.String clientID,
                                    java.lang.String queueName,
                                    long timeout)
                                    throws
javax.naming.NamingException,
                                           javax.jms.JMSException
```

抛出：

javax.naming.NamingException

javax.jms.JMSException

---

receive

```
public static java.util.List<javax.jms.TextMessage>
receive(javax.jms.Session session,
                                   java.lang.String fileID,
                                   java.lang.String batchID,
                                   java.lang.String clientID,
                                   java.lang.String queueName,
                                   long timeout)
                                   throws javax.jms.JMSException
```

接收回执

参数：

session - 由外部程序关闭

fileID -

clientID -

queueName -

timeout -

返回：

抛出：

javax.jms.JMSException

---

receive

```
public static void receive(java.lang.String  url,
                           java.lang.String  fileID,
                           java.lang.String  clientID,
                           java.lang.String  batchID,
                           java.lang.String  queueName,
                           long  timeout,
                           java.lang.String  bakMsgPath)
                    throws java.io.FileNotFoundException,
                           javax.xml.stream.XMLStreamException,
                           javax.xml.stream.FactoryConfigurationError,
                           javax.jms.JMSException,
                           javax.naming.NamingException,
                           java.io.UnsupportedEncodingException
```

将接收到的回执拼成一个文档

参数：

url -

filePath -

queueName -

timeout -

bakMsgPath -

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

java.io.UnsupportedEncodingException

---

receive

```
public static void receive(java.lang.String  url,
                           java.lang.String  fileID,
                           java.lang.String  clientID,
                           java.lang.String  batchID,
                           java.lang.String  queueName,
                           long  timeout,
                           javax.xml.stream.XMLStreamWriter  writer)
                    throws java.io.FileNotFoundException,
                           javax.xml.stream.XMLStreamException,
                           javax.xml.stream.FactoryConfigurationError,
                           javax.jms.JMSException,
                           javax.naming.NamingException
```

将接收到的回执拼成一个文档

参数：

url -

fileID -

queueName -

timeout -

writer -

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

---

receiveWithDoc

```
public static org.w3c.dom.Document receiveWithDoc(java.lang.String  url,
                                                   java.lang.String  fileID,
                                                   java.lang.String  clientID,
                                                   java.lang.String  batchID,
                                                   java.lang.String  queueName,
                                                   long  timeout)
                                            throws java.io.FileNotFoundException,
                                                   javax.xml.stream.XMLStreamException,
                                                   javax.xml.stream.FactoryConfigurationError,
                                                   javax.jms.JMSException,
                                                   javax.xml.parsers.ParserConfigurationException,
                                                   javax.naming.NamingException
```

将接收到的回执拼成一个文档

参数：

url -

fileID -

queueName -

timeout -

返回：

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.xml.parsers.ParserConfigurationException

javax.naming.NamingException

---

receiveAndWriteDoc

```
public static void receiveAndWriteDoc(java.lang.String  url,
                                       java.lang.String  batchID,
                                       java.lang.String  fileID,
                                       java.lang.String  clientID,
                                       java.lang.String  queueName,
                                       long  timeout,
                                       java.lang.String  docPath,
                                       java.lang.String  charset)
                                throws javax.naming.NamingException,
                                       javax.jms.JMSException,
                                       java.io.UnsupportedEncodingException,
                                       java.io.FileNotFoundException,
                                       javax.xml.stream.XMLStreamException,
                                       
javax.xml.stream.FactoryConfigurationError
```

抛出：

javax.naming.NamingException

javax.jms.JMSException

java.io.UnsupportedEncodingException

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

---

receiveAndWriteDoc

```
public static void receiveAndWriteDoc(javax.jms.Connection  conn,
                                       java.lang.String  batchID,
                                       java.lang.String  fileID,
                                       java.lang.String  clientID,
                                       java.lang.String  queueName,
                                       long  timeout,
                                       java.lang.String  docPath,
                                       java.lang.String  charset)
                                throws javax.naming.NamingException,
                                       javax.jms.JMSException,
                                       java.io.UnsupportedEncodingException,
                                       java.io.FileNotFoundException,
                                       javax.xml.stream.XMLStreamException,
                                       
javax.xml.stream.FactoryConfigurationError
```

抛出：

javax.naming.NamingException

javax.jms.JMSException

java.io.UnsupportedEncodingException

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

---

receiveAndWriteDoc

```
public static void receiveAndWriteDoc(javax.jms.Session  session,
                                       java.lang.String  fileID,
```

```
                                        java.lang.String  batchID,
                                        java.lang.String  clientID,
                                        java.lang.String  queueName,
                                        long  timeout,
                                        java.lang.String  docPath,
                                        java.lang.String  charset)
                                throws javax.jms.JMSException,
                                        javax.xml.stream.XMLStreamException,
                                        java.io.UnsupportedEncodingException,
                                        java.io.FileNotFoundException,
                                        javax.xml.stream.FactoryConfigurationError
```

抛出：

```
javax.jms.JMSException
javax.xml.stream.XMLStreamException
java.io.UnsupportedEncodingException
java.io.FileNotFoundException
javax.xml.stream.FactoryConfigurationError
```

```
receiveWithCountAndWriteDoc
public static void receiveWithCountAndWriteDoc(javax.jms.Connection  conn,
                                        java.lang.String  batchID,
                                        
```

```
java.lang.String  fileID,
                                                    
java.lang.String  clientID,
                                                    
java.lang.String  queueName,
                                                        long  count,
                                                    
java.lang.String  docPath,
                                                    
java.lang.String  charset)
                                          throws
javax.naming.NamingException,
                                                javax.jms.JMSException,
                                                    
java.io.UnsupportedEncodingException,
                                                    
java.io.FileNotFoundException,
                                                    
javax.xml.stream.XMLStreamException,
                                                    
javax.xml.stream.FactoryConfigurationError
```

抛出：

```
javax.naming.NamingException
javax.jms.JMSException
java.io.UnsupportedEncodingException
java.io.FileNotFoundException
javax.xml.stream.XMLStreamException
javax.xml.stream.FactoryConfigurationError
```

receiveWithCountAndWriteDoc

public static void receiveWithCountAndWriteDoc(javax.jms.Session session,

java.lang.String fileID,

java.lang.String batchID,

java.lang.String clientID,

java.lang.String queueName,

long count,

java.lang.String docPath,

java.lang.String charset)

throws javax.jms.JMSException,

javax.xml.stream.XMLStreamException,

java.io.UnsupportedEncodingException,

java.io.FileNotFoundException,

javax.xml.stream.FactoryConfigurationError

抛出：

javax.jms.JMSException

javax.xml.stream.XMLStreamException

java.io.UnsupportedEncodingException

java.io.FileNotFoundException

javax.xml.stream.FactoryConfigurationError

---

receiveWithCount

```
public static java.util.List<javax.jms.TextMessage>
receiveWithCount(
    javax.jms.Session session,
                     java.lang.String fileID,
                     java.lang.String batchID,
                     java.lang.String clientID,
                     java.lang.String queueName,
                     long count)
                     throws javax.jms.JMSException
```

跟数量接收消息,接收回执,一直到消息接收完毕

参数:

session - 由外部程序关闭

fileID -

clientID -

queueName -

timeout -

返回:

抛出:

javax.jms.JMSException

---

receiveWithCount

```
public static java.util.List<javax.jms.TextMessage>
receiveWithCount(java.lang.String url,
                 java.lang.String clientID,
```

java.lang.String batchID,

java.lang.String fileID,

java.lang.String queueName,

long count)

throws

javax.naming.NamingException,

javax.jms.JMSException

接收回执

参数:

url -

filePath -

queueName -

count -

返回:

抛出:

javax.naming.NamingException

javax.jms.JMSException

---

receiveWithCount

public static java.util.List<javax.jms.TextMessage>

receiveWithCount(

javax.jms.Connection conn,

java.lang.String clientID,

java.lang.String batchID,

java.lang.String fileID,

java.lang.String queueName,

long count)

throws

javax.naming.NamingException,

javax.jms.JMSException

抛出：

javax.naming.NamingException

javax.jms.JMSException

---

receiveWithCount

public static void receiveWithCount(java.lang.String url,

java.lang.String clientID,

java.lang.String batchID,

java.lang.String fileID,

java.lang.String queueName,

long count,

java.lang.String bakMsgPath)

throws

java.io.FileNotFoundException,

javax.xml.stream.XMLStreamException,

javax.xml.stream.FactoryConfigurationError,

javax.jms.JMSException,

javax.naming.NamingException,

java.io.UnsupportedEncodingException

将接收到的回执拼成一个文档

参数：

url -

fileID -

queueName -

count -

bakMsgPath -

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

java.io.UnsupportedEncodingException

---

receiveWithCount

```
public static void receiveWithCount(java.lang.String  url,
                                     java.lang.String  clientID,
                                     java.lang.String  batchID,
                                     java.lang.String  fileID,
                                     java.lang.String  queueName,
                                     long  count,
                                     java.lang.String  bakMsgPath,
                                     java.lang.String  charset)
                              throws
```

java.io.FileNotFoundException,

javax.xml.stream.XMLStreamException,

javax.xml.stream.FactoryConfigurationError,

javax.jms.JMSException,

javax.naming.NamingException,

java.io.UnsupportedEncodingException

将接收到的回执拼成一个文档

参数:

url -

fileID -

queueName -

count -

bakMsgPath -

抛出:

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

java.io.UnsupportedEncodingException

---

receiveWithCount

```
public static void receiveWithCount(javax.jms.Connection  conn,
                                     java.lang.String  clientID,
                                     java.lang.String  batchID,
                                     java.lang.String  fileID,
                                     java.lang.String  queueName,
                                     long  count,
```

```
                                        java.lang.String  bakMsgPath,
                                        java.lang.String  charset)
                                 throws
java.io.FileNotFoundException,
                                        
javax.xml.stream.XMLStreamException,
                                        
javax.xml.stream.FactoryConfigurationError,
                                        javax.jms.JMSException,
                                        javax.naming.NamingException,
                                        
java.io.UnsupportedEncodingException
```

将接收到的回执拼成一个文档

参数：

url -

fileID -

queueName -

count -

bakMsgPath -

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

java.io.UnsupportedEncodingException

---

receiveWithCount

```
public static void receiveWithCount(java.lang.String  url,
```

```
                                        java.lang.String  fileID,
                                        java.lang.String  batchID,
                                        java.lang.String  clientID,
                                        java.lang.String  queueName,
                                        long  count,

javax.xml.stream.XMLStreamWriter  writer)
                                    throws
java.io.FileNotFoundException,

javax.xml.stream.XMLStreamException,

javax.xml.stream.FactoryConfigurationError,
                                        javax.jms.JMSException,
                                        javax.naming.NamingException
```

将接收到的回执拼成一个文档

参数：

```
url -
filePath -
queueName -
count -
writer -
```

抛出：

```
java.io.FileNotFoundException
javax.xml.stream.XMLStreamException
javax.xml.stream.FactoryConfigurationError
javax.jms.JMSException
javax.naming.NamingException
```

receiveWithCount

```
public static void receiveWithCount(javax.jms.Connection  conn,
                                     java.lang.String  fileID,
                                     java.lang.String  batchID,
                                     java.lang.String  clientID,
                                     java.lang.String  queueName,
                                     long  count,
                                     javax.xml.stream.XMLStreamWriter  writer)
                                     throws java.io.FileNotFoundException,
                                     javax.xml.stream.XMLStreamException,
                                     javax.xml.stream.FactoryConfigurationError,
                                     javax.jms.JMSException,
                                     javax.naming.NamingException
```

将接收到的回执拼成一个文档

参数：

url -

filePath -

queueName -

count -

writer -

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.naming.NamingException

---

receiveWithCountAndDoc

public static org.w3c.dom.Document receiveWithCountAndDoc(java.lang.String url,
java.lang.String batchID,
java.lang.String clientID,
java.lang.String fileID,
java.lang.String queueName,
long count)
throws java.io.FileNotFoundException,
javax.xml.stream.XMLStreamException,
javax.xml.stream.FactoryConfigurationError,
javax.jms.JMSException,
javax.xml.parsers.ParserConfigurationException,
javax.naming.NamingException

将接收到的回执拼成一个文档

参数：

url -

fileID -

queueName -

count -

返回：

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.xml.parsers.ParserConfigurationException

javax.naming.NamingException

---

receiveWithCountAndDoc

```
public static org.w3c.dom.Document
receiveWithCountAndDoc(javax.jms.Connection conn,
                       java.lang.String batchID,
                       java.lang.String clientID,
                       java.lang.String fileID,
                       java.lang.String queueName,
                       long count)
                throws
java.io.FileNotFoundException,
javax.xml.stream.XMLStreamException,
javax.xml.stream.FactoryConfigurationError,
                       javax.jms.JMSException,
javax.xml.parsers.ParserConfigurationException,
                       javax.naming.NamingException
```

抛出：

java.io.FileNotFoundException

javax.xml.stream.XMLStreamException

javax.xml.stream.FactoryConfigurationError

javax.jms.JMSException

javax.xml.parsers.ParserConfigurationException

javax.naming.NamingException

---

getHostName

public static java.lang.String getHostName()

---

createMessageConsumer

public static javax.jms.MessageConsumer createMessageConsumer(

javax.jms.Session session,

java.lang.String queueName,

java.lang.String fileID,

java.lang.String clientID,

java.lang.String batchID)

throws javax.jms.JMSException

抛出：

javax.jms.JMSException

---

createMessgeSelector

public static java.lang.String createMessgeSelector(java.lang.String fileID,

java.lang.String clientID,

java.lang.String batchID)

8.3.2. JMS 实例代码

(1) 发送文件

```
public void testSendFile() throws Exception{
    Map<String, String> object = new HashMap<String, String>();
    object.put("account", "design");
    object.put("sender", "1101");
    object.put("receiver", "0001");
    JMSUtil.sendFile("tcp://localhost:61616",
"C:\\atemp\\h\\InventoryBaseFile.xml",                    "UAP_EAI_MSG_Q",
"UAP_EAI_MSG_RET_Q", object);
}
```

(2) 接收回执

```
public void testReceiver() throws Exception{
    JMSUtil.receive("tcp://localhost:61616",
"C:\\atemp\\h\\InventoryBaseFile.xml",    "UAP_EAI_MSG_RET_Q",    1000,
"C:\\atemp\\h\\abcd.bak.xml");
}
```

8.4. 大文件传输模式

(1) 参数设置，选择大文件传输模式

(2) 参数设置，保留大文件传输模式临时文件

在传输大文件的时候，选中该选项，可以在后台 pfxx\pfxxtemp_work_temp_目录下查看导入的大文件是否正确被传送到后台。

(3) 手工加载界面

待发送文件，只能发送一次

刷新，重置已发送文件记录

大文件传输模式，文件传输到后台，即返回回执，请到交换平台日志节点中查看日志

(4) 可行方案

设置，选择大文件传输模式

设置，单据导入规则选择 同一文档里的每张单据单独处理

设置，JMS/后台线程数，请根据后台服务器性能选择合适值（重启 中间生效）建议了不要超过 10 个

将大文件拆分为多个文件，其中每个文件不要超过 100M，可同时多个 客户端发送。建议了不要超过 10 个.

平台日志节点中查看日志

9. 附录

9.1. 发送结果错误码

1 正常处理完毕，没有错误

-10000:第一类：环境及未知错误：外部环境、网络错误、未知异常（需要检查外部环境、网络设置、端口等等）

-11000 Servlet 处理异常

-12001 数据库链接错误

-20000:第二类：实施配置错误：信息交换平台配置错误(总体参数、外部系统、帐套、辅助信息、校验文件等等配置错误)

-21000 整体配置错误

-21001 没有设置默认帐套

-21002 初始化数据源错误

-21101 无法找到相应的校验文件

-21102 无法找到单据配置文件

-21103 无法找到插件注册文件

-21104 无法找到 SENDURL 文件

-21105 无法找到辅助信息注册文件

-21106 文件超过最大单篇传输上限

-22000 与单据相关配置

-22001 无法找到对应的注册插件

-22002 无法找到插件类、实例化业务插件出现错误

-22021 档案只可以导入到集团，不能导入到公司

-22022 档案只可以导入到公司，不能导入到集团

-22030 待分配公司编码不正确!非法公司编码

-22100 XML 转换为 VO 时出现错误

-30000 第三类：用户数据错误，主要包括以下两种:

3.1 导入数据时出现基本校验错误，这些错误由信息交换平台校验发现（重复导入、格式不对、信息不全、字段类型错误、无法翻译基础数据等等）

-31000 单据重复导入错误

-31001 删除不存在的单据错误

-31002 更新不存在的单据错误

-31003 所导入的 XML 文件格式不正确

-31004 单据本身没有错误，但是由于配置为大事务，受其他单据影响而无法导入

-31100 无效操作类型

-31101 无效的接收公司

-31102 无效的发送方

-31113 无效的单据类型

-31114 根据帐套无法找到数据源

-31115 根据公司无法取得默认主体帐簿

-31116 外部系统已经停用，不能导入

-31117 无效的主体帐簿

-31118 无效的科目方案

-31119 无效的 url 格式 应为<公司编码@主体账簿编码>格式

-31200 字段内容错误－不能为空，类型错误，无法翻译等

-31201 发送方地址不合法

-31201 记录内容错误－没有为特定表记录类型定义特定属性等

-31202 单据转换翻译错误

-31203 单据导入环境初始化异常

-31204 基础档案资源列表里没有名为[{0}]的档案，请在自定义档案里添加！

-31205 往公司发送时，pk_corp=0001 不合法

-31206 根据基本档案[{0}]无法翻译[{1}]字段,待翻译值:{2},翻译方式:{3}.

-31207 组织字段不一致

3.2 所导入的数据出现业务错误，这些错误与实际的单据业务相关，由业务插件发现。

-32000 业务插件处理错误

-32001 单据编码重复错误

-32002 名称重复错误

总账业务插件处理错误返回值：(兼容老的错误编码)

错误编码：	错误说明：
-5910001	"凭证借贷金额不平！"
-5910003	"凭证没有分录！"
-5910004	"日期在系统期间没有定义！"
-5910005	"总账系统尚未启用！"
-5910006	"日期早于总账的启用期间！"
-5910007	"日期所在期间已结账！"
-5910008	"使用的凭证号已存在，不允许重复！"
-5910009	"未录入凭证类别！"
-5910010	"使用的凭证类别未定义！"
-5910011	"凭证分类没有设置！"
-5910012	"凭证类别定义的科目限制类型有误！"
-5910013	"会计年度不合法！"
-5910014	"摘要为空！"
-5910015	"科目为空！"
-5910016	"本币金额为"0"！"
-5910017	"数量、金额都为"0"！"
-5910018	"币种为空！"
-5910019	"科目的受控系统与制单系统不匹配！"
-5910020	"科目不是末级科目。"
-5910021	"科目已被封存。"
-5910022	"辅助核算未录！该科目已设定辅助核算，辅助核算

不可以为空。"

-5910023 "辅助核算类型与科目的设置不符，请重新录入辅助核算。"

-5910024 "数据库辅助核算表的记录有错！"

-5910025 "辅助核算有错！该科目未设定辅助核算，不允许录入辅助核算。"

-5910026 "科目是借方科目且设置了发生额控制!不允许录入贷方金额。"

-5910027 "科目是贷方科目且设置了发生额控制!不允许录入借方金额。"

-5910028 "本凭证类别已经设置了借方必有科目，而凭证中没有引用该科目，请检查借方科目！"

-5910029 "本凭证类别已经设置了贷方必有科目，而凭证中没有引用该科目，请检查贷方科目！"

-5910030 "本凭证类别已经设置了凭证必有科目，而凭证中没有引用该科目，请检查科目！"

-5910031 "本凭证类别已经设置了凭证必无科目，而凭证中引用了该科目，请检查科目！"

-5910032 "未定义的科目限制类型！"

-5910033 "会计期间不合法！"

-5910034 "凭证已被删除。"

-5910035 "凭证的原始数据已被破坏，无法完成操作。"

-5910036 "凭证已被记账，不能删除。"

-5910037 "凭证已被审核，不能删除。"

-5910038 "凭证已被签字，不能删除。"

-5910039 "凭证已有分录被勾对，不能删除。"

-5910040 "凭证被设定只能由本人修改，您无权删除别人的凭证。"

-5910041 "有其他用户在操作，请稍候再试。"

-5910042 "错误或无效的凭证号。"

-5910043 "正在修改的凭证已被别人作废。无法保存数据。"

-5910044 "科目信息有错！"

-5910045 "错误的科目公司引用"

-5910046 "凭证日期与会计年度、期间不匹配。"

-5910047 "凭证类别所在公司与制单公司不符。"

-5910048 "科目设置了余额控制，余额不允许为负。"

-5920001 "原币为"0"。"

-5920002 "原币合计不平。"

-5920003 "辅币合计不平。"

-5920004 "数量为"0"。"

-5990001 "凭证的辅助信息配置错误，不能有空项"

-5990002 "凭证辅助信息配置系统来源或凭证类别没有匹配项"

-5990003 "凭证辅助信息配置的修改删除控制字符串长度有错"

-5990004 "外部系统试图修改不存在的凭证"

-5990005 "外部系统试图删除不存在的凭证"

-5990006 "不能从总账系统中删除从外系统导入的凭证"

-5990007 "物料生产档案的辅助信息配置错误,库存组织编码和名称不能有空项"

-5990008 料生产档案的辅助信息配置错误，库存组织编码应该是形如'001'的三位数"

-40000 其他未知错误

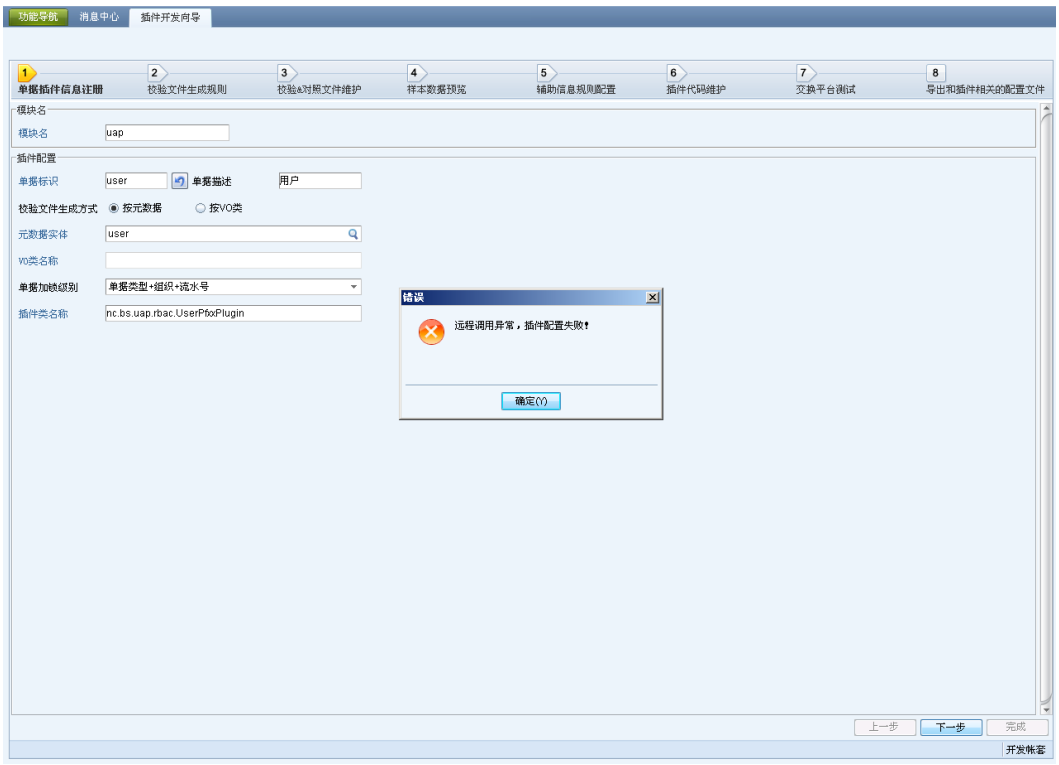
9.2. K 系统自定义项目档案样本 defdoc.xml

```
<?xml version="1.0" encoding='GB2312'?>
<ufinterface account="test" billtype="defdoc" filename="defdoc.xml" isexchange="Y" proc="add" receiver="yk"
```

```
replace="Y" roottag="" sender="1101">
    <defdoc id="12d8">
        <!--自定义档案内容编码-->
        <code>01d2a</code>
        <!--自定义档案内容名称-->
        <name>意外险</name>
        <!--自定义档案内容系统属性，可不用填写-->
        <docsystype></docsystype>
        <!--所属公司-->
        <company>0001</company>
        <!--所属定义档案,要求用名称，否则就得修改校验文件里该字段
的导入公式-->
        <defdoc>保险类别</defdoc>
        <!--封存标志-->
        <sealflag>Y</sealflag>
    </defdoc>
</ufinterface>
```

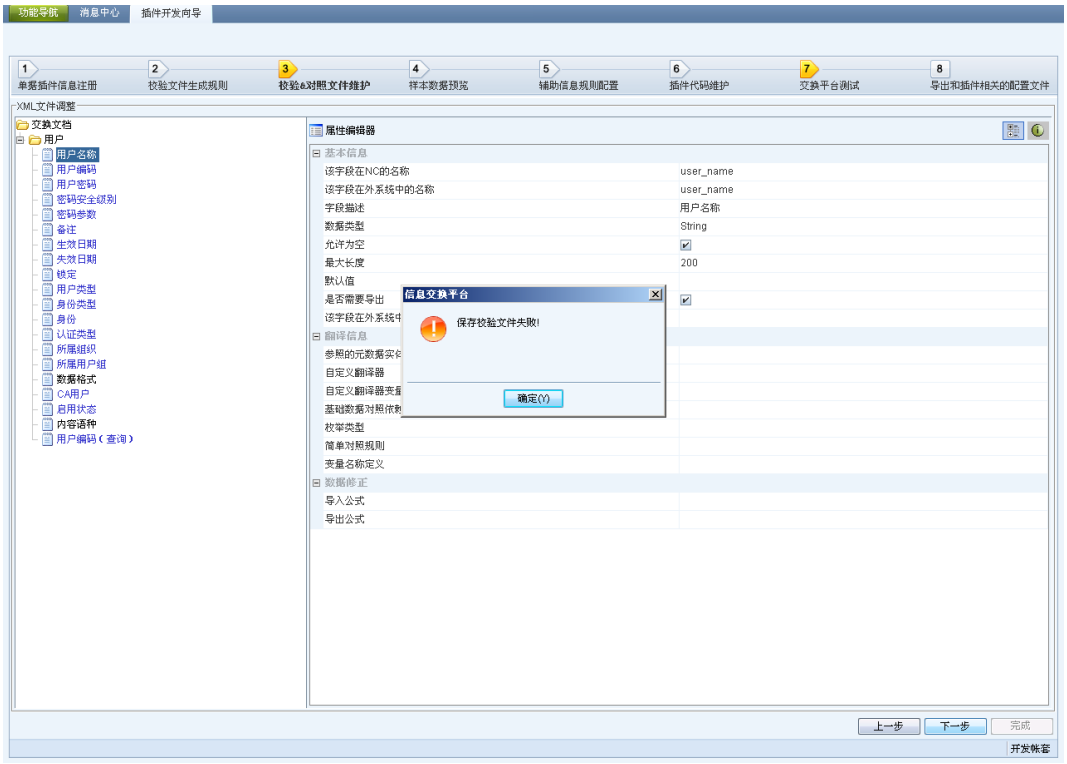
10. 常见问题

问题 1:“插件开发向导”点击下一步时提示“远程调用异常，插件配置失败!”，如图



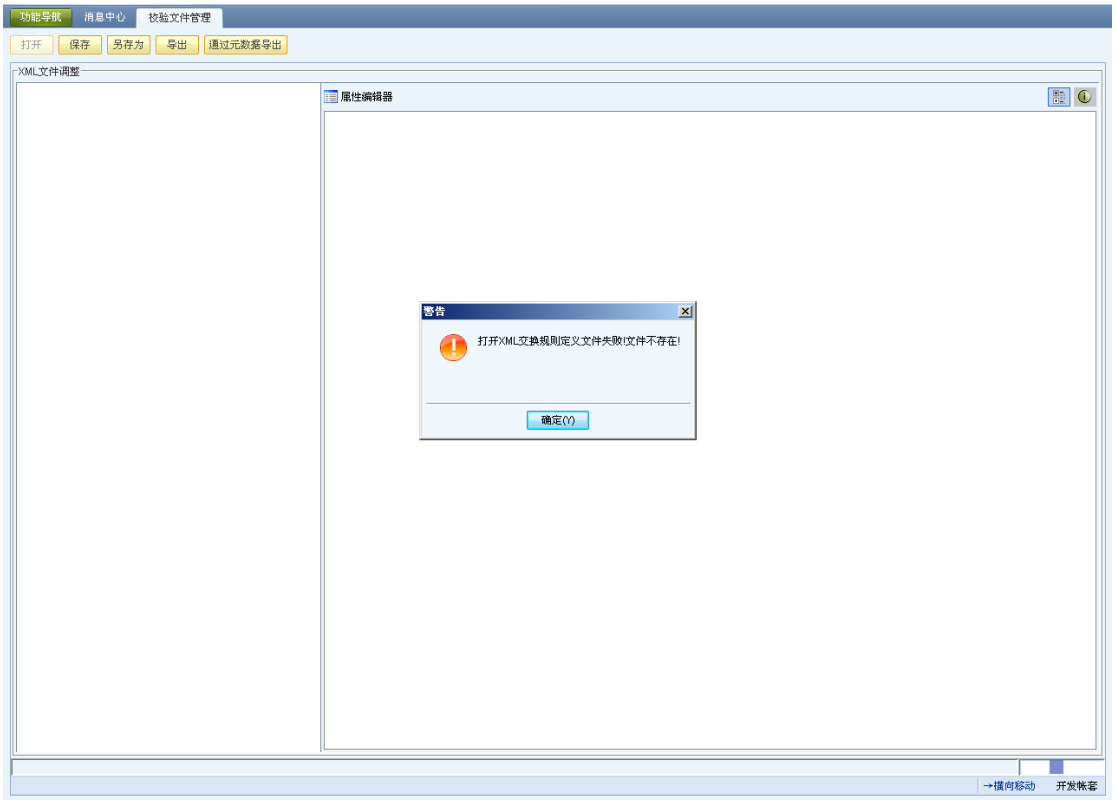
分析解决：有可能是 NCHOME\pfxx\文件夹或文件夹下的文件是只读的，请修改文件夹或文件为可读写。

问题 2:“插件开发向导”的第三步“校验&对照文件维护”点击下一步时提示“保存校验文件失败!”，如图



分析解决：同问题 1 的解决方法，将 NCHOME\pfxx\文件夹或文件夹下的文件改为可读写。

问题 3：“校验文件管理”选择了单据类型，无法打开文件，提示“打开 XML 文件规则定义文件失败！文件不存在！”，如图



分析解决：请确认 NCHOME\pfxx\billdefine\uap(按模块划分，根据模块不同，该文件夹不同)下是否存在对应单据的文件。

问题 4：“手动加载界面”发送数据以后返回的回执文件提示“从后台读取交换规则定义文件发生错误，请检查文件是否存在！”，如图

分析解决：同问题 3，请确认 NCHOME\pfxx\billdefine\uap(按模块划分，根据模块不同，该文件夹不同)下是否存在对应单据的文件。