**交换池系统**

**总体设计**

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 版本 | 提交人 | 备注 |
| 2014-11-21 | **1.0** | **Wangm** |  |
|  |  |  |  |
|  |  |  |  |

# 设计原则

1. 尽可能采用开源项目
2. 该交换池系统存在两大难点，一是系统本身存在大量的IO操作，来源于大量数据文件的传输与存储;二是适配来自多个业务系统的多种异构数据，而且异构数据有可能是由来自多个数据源的数据组合。在下列章节做系统设计时会主要针对这两点提供解决方案。

# 总体架构

## 总体架构图



**说明 ：**

上图描述了交换池系统的整体架构图，描述如下：

1. 前置节点与交换池中心之间实现控制流/信息流与数据流分离的原则，前置节点与交换池中心的控制端之间通过单向心跳的方式交换控制流/信息流；前置节点与交换池中心的数据交换代理间交换数据流。
2. 前置节点与业务系统之间通过适配的方式来交换数据，适配方式可通过交换池中心配置扩展。
3. 为缓解交换池中心作为数据文件中转的IO压力，如果两前置节点之间网络条件良好并且交换池中心许可的情况下，两前置节点间可直接交换数据，为安全考虑，两前置节点间仅可通过下载的方式交换数据。
4. 前置节点本地设置交换仓储、数据仓储、预览仓储
   1. 交换仓储：用以存放数据交换的临时文件（待发送文件、接收中数据文件、打包/解包数据文件）。
   2. 数据仓储：用以存放转换后的OSRE数据文件及自定义数据文件，用户需要在数据仓储中建立子仓储存放数据文件。
   3. 预览数据仓储：如果需要对数据仓储中的某些数据文件进行预览，需要对数据文件进行预处理，处理到预览数据仓储中进行预览。

## 交换池中心

## 前置节点

1. **前置节点与业务系统集成**

前置节点通过File、FTP、DB方式从业务系统获取数据，获取操作通过连接适配器+配置文件来实现，详见：\*\*\*\*

1. **前置节点数据存储**

前置节点数据存储分为三类：

1. 交换仓储：用于存放已打包的待发送数据文件；已接收的打包数据文件
2. 数据仓储：已转换的OSRE存储格式的数据文件，数据仓储下可以建立子仓储。
3. 预览数据仓储：用于预览的数据仓储，用MongoDB存储，如果某批数据转换成OSRE格式后需要预览，可以定义任务将元数据提取到MongoDB中，进行检索预览；可默认将所有转换后的OSRE元数据放入MongoDB中，记录数据文件ID或数据文件批次，用于检索。
4. **前置节点间数据交换**

前置节点间数据交换时，会先由数据目标前置节点A发起到数据提供前置节点B的网络连接情况探测，如果网络条件良好，则A创建一个临时的安全认证下载链接，通过交换中心交付给B，由B直接从A下载数据文件到本地，具体流程见：\*\*\*\*\*\*。

## WEB端及移动端

1. **展现**

交换池中心界面操作的一部分服务调用来自于相应的前置节点。



前置节点在注册的时候提供一个服务器IP，用户登录交换中心用户操作界面通过该IP来跨域调取对前置节点上的数据文件及日志文件的预览或检索功能。

注：如提供的服务器IP为内网IP，则用户处在内网环境下才能使用这部分服务。

1. **移动端开发**
   1. **响应式页面开发**

可应用组件：bootstrap等

容器：浏览器或者APP内嵌浏览器展现方式

优点：只需应用响应式框架开发响应式页面即可，不用额外开发APP。

缺点：在移动端组织页面及调用移动设备接口时不如APP灵活，特别是有些设备接口不允许调用。

* 1. **APP方式开发**

Android/ios

## Bootstrap开发简介

Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口（viewport）尺寸的增加，系统会自动分为最多12列。如果一“行（row）”中包含了的“列（column）”大于 12，多余的“列（column）”所在的元素将被作为一个整体另起一行排列。



示例如下：

<div class="row">

<div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12 .col-sm-6 .col-md-8</div>

<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>

</div>

<div class="row">

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

<!-- Optional: clear the XS cols if their content doesn't match in height -->

<div class="clearfix visible-xs-block"></div>

<div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>

</div>

# 交换池中心

交换池中心由处理控制流/信息流的交换池控制端和处理数据流的数据存储组成。

## 交换池控制端

交换池控制端用于整个交换池体系的注册、配置、调度，所有的操作都面向交换池控制端进行管理，交换池控制端连接MySQL数据库进行数据存储。

### 功能框架



#### 注册管理

1. 机构注册
2. 前置节点注册
3. 前置节点安装包及版本注册
4. 适配器注册
5. 用户注册

#### 任务管理

1. 导入任务
2. 导出任务
3. 数据交换任务
4. 发布/订阅
5. 日志管理

#### 数据文件管理

1. 仓储管理
2. 数据文件管理

#### 控制端服务

1. 心跳处理
2. 安全管理
3. 任务组装
4. 任务调度
5. 任务监控
6. 适配器发布

### 多机负载均衡



1. Nginx：交换中心主要交换信息流和控制流，数据相对来说比较小，在负载均衡10台以下规模的交换中心，使用单台nginx做负载均衡即可,可使用nginx的ip\_hash/upstream\_hash做基于session一直的负载均衡。
2. Memcached/Radis：如果交换中心有其他的全局共享缓存数据，则需要放置于memcached/Radis中,其中需要共享缓存的数据
   1. 用户Session信息
   2. 正在执行的任务信息
   3. 正在交换的数据信息
3. Mysql Cluster：依据实际开通的机构及数据量，可选择采用单mysql数据库/mysql集群部署。

## 数据交换

数据交换代理主要用于：面向客户端交换数据、客户端权限验证、断点续传协议支持及交换数据缓存，可采用下面两种方式搭建：

1. **代理（nginx）+分布式存储（Hadoop HDFS）**

**方案：**

Nginx用于数据交换的代理，效率很高，可面向客户端设置多个nginx代理用于数据流的交换，在交换中心给各前置节点授权上传数据时同时指定其中一个nginx。Nginx后接统一的HDFS分布式文件系统用于数据文件存储。

**优点：**

Nginx、HDFS都是开源界千锤百炼的项目，成熟度及效率都有保障，有成熟的社区支持。

**缺点：**

需要在nginx上进行二次开发，提供权限校验及断点续传功能。

1. **OSS系统**

**方案：**

使用OSS系统来进行数据流的交换，交换池中心代替前置节点获取上传/下载数据文件的签名，并为前置节点指定OSS系统用来交换数据文件的datanode。

**优点：**

OSS已经实现了权限管理、负载均衡及断点续传功能；暂时未实现数据文件的备份，可用datanode连接HDFS来存储数据文件进行备份。

**缺点：**

性能及稳定性上赶不上Nginx。

# 前置节点

# 适配器



# 系统流程

## 心跳

1. **心跳模型**

前置节点周期性地向交换中心发起心跳调用，将前置节点状态及任务运行状态等信息汇报给交换中心，并从交换中心获取命令及配置信息。

心跳周期：心跳之间的间隔时间，可配置。

1. **心跳安全**

前置节点汇报时，需要提供对关键信息的签名，签名使用前置节点在交换中心注册时分发的KEY作为签名种子，验证签名并且验证前置节点IP。

签名关键信息：前置节点ID、汇报时间等，其他信息根据需要添加

签名过期：汇报时间超过3次心跳周期，即判定为过期，此次心跳不接收，不分配命令。

1. **在线状态判定**

前置节点超过3次心跳周期没有汇报，则判定离线。

1. **心跳交换信息**
2. 汇报信息：
   1. 前置节点运行健康状况汇报
   2. 数据处理任务运行状况汇报
   3. 前置节点到交换中心/数据源前置节点网络状况汇报
   4. 任务命令确定接收汇报
   5. 数据交换汇报
   6. 。。。。。。
3. 响应信息：
   1. 任务命令
   2. 任务结果确定接收命令
   3. 数据交换命令
   4. 。。。。。。

## 数据交换

数据交换流程序列图如下：



## 数据处理



说明：执行数据处理任务阶段分为：数据获取、数据转换（可选）、数据校验（可选）、数据导出。

1. 数据获取：从DB、FTP、File读取数据。
2. 如没有数据转换、校验阶段，则“数据获取”直接读取数据存放到“数据导出”指定的存储目录。
3. 如有数据转换、校验阶段，则“数据获取”会将数据（xml/excel/txt/db）读取到内存的DataRecord（见6.1）对象中，然后执行后续操作。
4. 数据转换：依据4.2格式转换定义，进行数据转换。
5. 数据校验：暂略
6. 数据导出：暂时提供基于OSRE导出，需制定导出仓储。

# 数据转换功能

## OSRE中间格式定义

OSRE中间格式在交换池系统中的配置如下图所示：



说明：

* 节点：以多级方式定义中间格式
* 名称：节点对应名称
* 必备性：节点是否必备
* 是否多值：节点是否可以多值
* 是否外联对象：该节点是否关联一个外联对象，针对外联对象的操作如下：
  1. 外联对象在存储时会被生成一个独立的xml数据文件进行存储，存储在配置的“外联对象文件存储相对路径”中。
  2. 在配置时外联对象可额外定义数据源。
  3. 一级节点默认可定义数据源
* 是否数字对象：表示该字段是否表示一个数字对象并且要实际获取该数字对象；该字段配置指导获取数字对象，存储到相应目录（“外联对象文件存储相对路径”指定），然后该字段生成针对该数字对象的对象描述。
* 外联对象文件存储相对路径：相对于本体目录的相对存储问题。

## 数据格式转入定义

### 配置数据源



### 配置转入规则



说明：

1. 一级节点默认需指定一个数据源，根据数据源解析源数据并放置于内存的DataRecord对象中，DataRecord对象定义见附录6.1，二级节点通过配置表达式从DataRecord中取值。
2. 如一个节点为外联节点（非数字对象），该节点可以沿用上级节点的DataRecord取值，则该节点的子节点通过配置表达式从上级节点的DataRecord取值；或者额外定义一个数据源，根据数据源解析数据并放置于内存的DataRecord对象中，则该节点的子节点通过配置表达式从该DataRecord取值。
3. 如一个节点为外联节点（是数字对象），则从数据源中获取数字对象存储，并在该节点生成数字对象关联。

## 数据格式转出定义

### 配置数据源

同4.2.1

### 配置转出规则



说明：

1. 指定数据源来指定转出数据存放位置。
2. 通过存储相对地址来定位生成文件存在位置。
3. 通过存储模板来定义生成文件样式。
4. 在上级节点指定的数据源配置可以获取下级节点的数据，指定方式为#{本节点代码.下级节点代码}，如果下级节点为多值，则将所有值拼装在一起，中间以逗号分隔。

## 数据处理任务配置



## 任务定义模板

1. **任务流程模板**

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

<workflow>

<globalProperties>

<property name="dataTypeId" value="$init\_dtype\_002"/>

<property name="dataTypeCode" value="book"/>

<property name="dataTypeVersion" value="1"/>

<property name="encoding" value="utf-8"/>

</globalProperties>

<datasources>

<datasource type="db"><!--数据获取方式 db/ftp/file/http-->

<adapter mainClass="" type="system"/>

<property name="" value=""/>

<property name="" value=""/>

</datasource>

<datasource type="ftp">

<adapter mainClass="" type="system"/>

<property name="" value=""/>

<property name="" value=""/>

</datasource>

</datasources>

<workflowNodes>

<workflowNode name="get">

<adapter id="init-adp-001" mainClass="" code="" type="plugin"/>

</workflowNode>

<workflowNode name="convert" dependOn="get">

<adapter id="init-adp-002" mainClass="" code="" type="plugin"/>

<import resource="convert.xml"/>

</workflowNode>

<workflowNode name="check" dependOn="convert">

<adapter mainClass="" type="system"/>

</workflowNode>

<workflowNode name="write" dependOn="check">

<adapter mainClass="" type="system"/>

</workflowNode>

</workflowNodes>

</workflow>

1. **转换映射模板**

对应于4.2.2

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

<DataRecord dataTypeCode="book" dataTypeVersion="1" refDataSource="1" target="select id,title creator from book" unzip="false" >

<DataField name="bookid" exp="@id"/>

<DataField name="title" exp="Substring(@title,0,20)"/>

<DataField name="creator" outline="true" digitalObj="false" refDataSource="2" target="/#{bookid}/creator.xml" unzip="false">

<DataField name="creatorid" exp="@creator.id"/>

<DataField name="name" exp="@creator.name"/>

</DataField>

<DataField name="Cover" outline="true" digitalObj="true" refDataSource="3" target="/#{bookid}/cover/\*.jpg" unzip="false">

<DataField name="creatorid" exp="@creator.id"/>

<DataField name="name" exp="@creator.name"/>

</DataField>

</DataRecord>

## 隐藏DB类数据导出模板(不转换成OSRE)

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

<DataRecord>

| <DataField name="" value=""/>

<DataField name="" value="">

<DataField name="" value=""/>

</DataField>

</DataRecord>

# 任务管理

## 任务定义

1. 导入任务
2. 导出任务
3. 传输任务

## 增量导入实现方式

通过为数据源建立增量索引的方式进行增量收割。

只为顶层数据源建立增量索引，其他数据源增量都由顶层数据源来关联。

1. **基于DB的增量方式**

DB数据源被收取记录拥有更新时间戳，任务每次运行成功后都更新收取起始时间戳为上一次任务收取截止时间戳。

1. **基于File及FTP的增量方式**

需要为数据源建立收取索引，以数据源文件的相对地址作为Key=MD5(相对地址)键索引，记录该文件是否被收割过。

## 内存数据

传输中的任务及数据文件状态由内存存储，每次系统重启由交换中心与前置节点交换回复中间状态。

# 关键表

# 附录

# DataRecord对象定义

**public** **class** DataRecord {

**public** **static** **final** String *METADATA\_ROOT\_NAME* = "root";

**private** String id;//记录ID

**private** **long** version;//记录的版本

**private** String source;//数据来源

**private** String via;//数据获取方式

**private** String batchId;//数据批号

**private** **long** createTime;//数据创建时间

**private** **int** level;//数据级别

**private** DataField metadata = **new** DataField(*METADATA\_ROOT\_NAME*, **null**);//元数据

**private** String schema;//元数据格式

**private** **long** schemaVersion;//元数据格式版本

**private** **long** position;//起始位置

**private** **int** length;//长度

**private** String orginMetadata;//源数据

**private** String orginMetadataFormat;//源数据文件格式

**private** String sourceDataFileURL;//来源数据文件相对路径

}

**public** **class** DataField{

/\*\*

\* 元数据项属性名

\*/

**private** String name;

/\*\*

\* 元数据项属性值

\*/

**private** String value;

/\*\*

\* 属性，使用于XML

\*/

**private** Map<String, String> attributes;

/\*\*

\* 子元数据项

\*/

**private** List<DataField> subFields;

}