**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。

## 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**

机构基本信息注册（字段见需求）

1. **前置节点注册**

前置节点主要注册信息如下表所示：

|  |  |
| --- | --- |
| 字段 | 说明 |
| 服务器对外IP | 用于交换中心的安全验证 |
| 服务器是否提供下载服务 | 用于前置节点间数据交换 |
| 服务器内网IP/外网独立IP | 用于本机构用户查阅前置节点数据及日志 |
| 关联安装版本 | 用于前置节点的版本管理，与适配器分发有关； |
| 是否在线 | 运行是否正常，这个状态根据心跳来判断，如果多次心跳间隔未收到心跳报告，则判定离线；离线的前置节点不分发任务。  该状态在内存中维护，交换池控制端启动时所有前置节点初始都设置为离线，待心跳来改变状态。 |
| 是否激活 | 人工设置该前置节点是否启用，不启用的前置节点不再接受新任务，但是会完成当前正在运行的任务。 |
| 最大并发任务 |  |
| Key | 每个前置节点分发一个KEY，用以在交换信息时，对关键信息进行签名的安全验证 |

1. **前置节点安装包及版本注册**

前置节点安装包及版本注册信息如下表所示：

|  |  |
| --- | --- |
| 字段 | 说明 |
| 版本 | Int数字表示 |
| 是否可用 | 安装了不可用版本的前置节点不再接收新任务，用以前置节点版本的平稳更新。 |
| 建议更新版本 |  |
| 下载链接 |  |

1. **适配器注册**

适配器主要注册信息如下，功能描述见[适配器](#_适配器)[。](#_Hlk404762073)

|  |  |
| --- | --- |
| 字段 | 说明 |
| 名称 | 类似：数据库连接 |
| 适配器类型 | 连接适配器（对应获取数据源类型）  导出适配器（对应存储数据源类型）  转换适配器  解析适配器  装配适配器  。。。。。。。。。。 |
| 适用最低版本 | 1.用户配置时根据用户选择的前置节点或者指定的前置节点版本来提供可选的适配器  2.为某前置节点定义任务指定运行流程时，需检查前置节点的版本与适配器版本，是否全部支持。  3.适配器开发保持向上兼容 |
| 适配器加载方式 | 可针对不同的安装版本指定不同的加载方式，主要是新发布的安装包会摘取一些适配器绑定到安装包中，这样运行起来比较稳定一些，管理也相对简单。  版本<=3,系统外加载，需要下载适配器。  版本>=4,系统内加载。 |
| 参数项 | 适配器所需参数项 |
| 依赖外部公共库 | 为了解决适配器加载依赖包冲突的问题，将外部公共库包统一管理，加载某适配器前先验证依赖包是否已加载，如果未加载，则下载到本地加载；以后系统每次重启直接将公共库加载，不需要重复下载。 |

1. **用户注册及权限管理**

用户及权限基本信息注册（字段见需求）。

#### 任务管理

1. **导入任务**

|  |  |
| --- | --- |
| 字段 | 说明 |
| 名称 |  |
| 指定导入流程/数据源 | 指定导入流程->导入流程包含（获取、转换）  转换流程中已指定数据源，配置数据源参数即可，校验可选  指定数据源->无转换/校验阶段 |
| 校验规则 | 从针对该文件格式的校验中选择一项 |
| 指定存储库区 | 在库区管理中配置，该任务每次运行时会在该库区下建立一个数据批号文件夹，本次导入数据以该文件夹为基准目录存放  数据批号格式待定（需满足全局唯一） |
| 任务属性 | |
| 是否启用 |  |
| 任务类型 | 一次性任务/周期性任务 |
| 启动时间 |  |
| 运行周期 |  |
| 获取方式 | 增量/全量，获取方式是针对数据格式的根来源定义的 |
| 数据起始时间 | 增量收割使用，但是不是所有来源的增量都可以直接用时间来衡量，有些需要做收取索引 |
| 任务上次开始时间 | 等待执行时的任务时间信息 |
| 任务上次结束时间 |
| 任务下次可运行时间 |
| 任务进度 |  |

注：

1. 可同时关联定义传输任务
2. 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。

1. **导出任务**

|  |  |
| --- | --- |
| 字段 | 说明 |
| 名称 |  |
| 指定导出流程/数据源 | 指定导出流程->导出流程包含（获取、转换）  转换流程中已指定数据源，配置数据源参数即可  指定数据源->无转换阶段 |
| 校验规则 | 从针对该文件格式的校验中选择一项 |
| 指定来源  库区/数据批号/数据筛选条件 |  |
| 任务属性 | |
| 是否启用 |  |
| 任务类型 | 一次性任务/触发性任务 |

1. **数据交换任务**

包括任务管理及交换数据文件管理

|  |  |
| --- | --- |
| 字段 | 说明 |
| 任务属性 | |
| 状态 |  |
|  |  |
| 交换数据文件属性 | |
| 进度 |  |
| 传输速率 |  |
| 交换数据文件描述 |  |
| 数据格式 |  |
| 数据批号 |  |
| 文件大小 |  |
| 状态 | 未传输/传输中/传输完毕 |
| 发送端 |  |
| 接收端 |  |

注：

1. 交换数据文件的部分信息（进度、传输速率）需要在内存中维护
2. 中断的传输任务在交换中心控制端启动时，重新发送命令
3. **发布/订阅**

确定是否需要后添加

1. **日志管理**

任务日志

数据处理/交换日志

#### 数据文件管理

1. 库区管理

基于前置节点指定的数据仓储基础上指定库区，库区分数据格式管理

即第一级为前置节点指定的数据仓储目录，第二级为数据格式，第三级为库区目录，第四级为数据批号，如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 前置节点指定的仓储目录 | | | |
|  | OSRE | | |
|  |  | 库区1 | |
|  |  |  | 批次m |
|  | 不分类 | | |
|  |  | 库区1 | |
|  |  |  | 批次n |
|  |  |  |  |

1. 数据管理

包括前置节点数据管理、交换池中心数据管理

|  |  |
| --- | --- |
| 字段 | 说明 |
| 数据ID |  |
| 数据批号 | 同一批数据在不同位置的存储，批号是相同的  因此在上传/下载某批次数据文件时可以先询问中心是否存在 |
| 归属  (某前置节点/中心) |  |
| 存储库区 |  |
| 数据格式 |  |
|  |  |

#### 文件格式管理

1. **文件格式定义**

* 文件格式图示：



* 文件格式字段说明：

|  |  |
| --- | --- |
| 字段 | 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。 |

* 文件格式存放

使用Json文件存储整个文件格式配置。

1. **转入模板定义**

* **图示：**



* **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[DataRecord对象定义](#_DataRecord对象定义)，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。

注：

1. ***系统初始化为OSRE格式建立转入模板,定义库区OSRE数据文件到DataRecord的映射,数据源指向库区数据批次。***
2. **转出模板**

* **图示：**



* 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。

注：

1. ***系统初始化为OSRE格式建立转出模板,定义DataRecord到库区OSRE数据文件的映射，数据源指向库区数据批次。***
2. 思考

在定义导入/导出模板时，可以采用两种方式：第一种方式如2-3节所示；第二种方式是先自定义一种格式M，然后完成M数据文件->DataRecord/ DataRecord->M数据文件的导入/导出模板，然后针对格式M和OSRE做映射。

#### 控制端服务

1. **心跳处理**

前置节点周期性地向中心控制端发起心跳调用，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。

* **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。

* **心跳交换信息**

1. 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
2. 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

* **心跳模型**

1. **安全管理**

* **账户安全**
* **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。

* **数据文件安全**

1. 对数据文件做完整性校验,数据文件发送前发送端对数据文件计算MD5摘要，该摘要发送给接收端，摘要要进行签名防伪造，接收端接收完毕也计算MD5摘要，两相对照，如果不相同则数据文件在发送过程中被损坏或篡改，可申请重新发送。
2. 对发送到中心的数据文件做自动备份。

* **安全通道**

Https访问

* 适配器发布安全

适配器的获取URL也需要签名，下载的安全包要计算MD5，防止伪造。

1. **任务组装**

根据[导入任务/导出任务](#_任务管理),构建任务描述模板，格式如下：

* **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>

* **转换映射模板**

<?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>

1. **任务调度**

* 系统配置两个阈值
  1. 前置节点数据处理任务最大并发数，包括导入/导出任务
  2. 交换池中心数据传输最大并发数
* 任务队列
  1. 控制端在内存中为每个在线的前置节点维护两个任务列表：运行中的任务、等待中的任务。
  2. 系统判定符合可执行条件的任务放在等待中的任务列表。
  3. 前置节点每次心跳汇报，控制端都检查该前置节点任务是否饱和，如果不饱和，则从等待中的任务中获取一个，以命令的形式放在给这个前置节点的响应中，并将该任务移到运行中任务列表。
  4. 如前置节点状态由在线->离线，则要清除内存中该前置节点的任务队列；反之，如状态由离线->在线，则从数据库中加载任务。
* 任务线程
  1. 前置节点状态监控线程：监控前置节点在线状态，如间隔三次心跳都没有收到心跳汇报，则判定离线。
  2. 前置节点任务监控线程：监控任务定义，如已符合执行条件，则将任务加入等待中任务列表。

1. **任务监控**

* 通过心跳汇报的方式实现，通过心跳获取任务执行、数据交换的相关信息。
* 监控信息除了最后的任务完成、传输完成汇报，其他的都是临时监控信息，不需要持久化，放置在内存中供用户查看即可。

1. **适配器发布**

适配器类及依赖包打成Zip包形式提供下载功能，下载适配器时需要提供前置节点ID、下载适配器ID，并签名

### 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

## 功能框架



### 心跳服务

前置节点周期性地向交换池中心控制端发起心跳请求。

### 仓储管理

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。

### 传输服务

1. **上传/下载服务**

* 通过HttpClient实现面向交换池中心/其他前置节点的数据服务。
* 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
* 数据交换前需要对传输的数据文件进行打包。

1. **数据文件安全下载服务**

数据文件安全下载链接格式如下：

http://#{前置节点地址}:#{前置节点端口}/download/#{数据批号}/#{签名},

交换中心收到安全下载链接时要验证签名，验证通过才发送给其他前置节点

### 任务调度

1. **任务模板解析**

解析**[任务组装](#_Hlk404932747" \s "1,5361,5366,179,,任务组装)**[的模板，按照模板指示调度适配器执行。](#_Hlk404932747" \s "1,5361,5366,179,,任务组装)

1. **适配器调度执行**

根据[适配器](#_适配器)章节说明，使用适配器加载服务加载并调度适配器运行，如适配器之间采用并行模式，适配器运行之间可以使用缓冲区，以批提交的方式提供速度。

### 适配器加载

### 任务监控

### 日志服务

### 预览服务

## 基于HTTP协议的断点续传协议

* 用于定义基于HTTP方式的断点续传
* 遵循HTTP标准协议(HTTP RFC2616)
* 当前只定义单线程断点续传
* 断点续传协议能够兼容正常非断点续传

### 下载断点续传定义

#### 流程定义

流程步骤说明如下：

1. 查看本地数据文件已下载数量，决定下一步从偏移点A开始下载
2. 发送下载偏移点A给服务端请求从A点下载。
3. 根据服务器端返回的偏移点B（B<=A）开始写入。

#### 协议定义

1. **请求**

* Header包括；

|  |  |  |
| --- | --- | --- |
| **字段名** | **是否必备** | **说明** |
| Range  (RFC 2616 (HTTP/1.1) Section 14.35) | 否 | 如果希望从断点续传，需设置该字段，示例如下：  Range:bytes=2000070-  或  Range:bytes=2000070-106786027  不设置该字段等同于：  Range:bytes=0- |

1. **响应**

* Header包括；

|  |  |  |
| --- | --- | --- |
| **字段名** | **是否必备** | **说明** |
| Content-Length | 是 | 内容长度 |
| Content-Encoding | 是 | 固定“UTF-8” |
| Content-MD5 | 否 |  |
| Content-Range  RFC 2616 (HTTP/1.1) Section 14.16 | 是 | 返回文件范围，示例如下：  bytes 500-1233/1234 (范围下载)  或  bytes 0-1233/1234 (类似全部下载) |
| Content-Disposition | 是 | 返回文件描述（文件名）  格式：  attachment; filename=${URLEncode(fileName,”UTF-8”)} |
| Content-Type | 是 | 媒体类型 |

* 状态码(Status-Code)

200：全部下载成功

206：局部下载成功，可能包括全部下载（bytes 0-1233/1234）

其他：错误

注：下载实际的写入偏移点由服务器返回的Content-Range字段指定。Content-Range起始点<=Range起始点

### 上传断点续传定义

#### 流程定义



#### 协议定义

1. **断点上传查询**
2. 请求:询问服务器端某数据文件已接收情况。
3. 响应：

* Header包括；

|  |  |  |
| --- | --- | --- |
| **字段名** | **是否必备** | **说明** |
| Range  (RFC 2616 (HTTP/1.1) Section 14.35) | 否 | 如果希望从断点续传，需设置该字段，示例如下：  Range:bytes=2000070-  或  Range:bytes=2000070-106786027  不设置该字段等同于：  Range:bytes=0- |

* 状态码(Status-Code)

200：响应成功

其他：错误

1. **断点上传**
   1. 请求

* Header:

|  |  |  |
| --- | --- | --- |
| **字段名** | **是否必备** | **说明** |
| Content-Length | 是 | 内容长度 |
| Content-Encoding | 是 | 固定“UTF-8” |
| Content-MD5 | 否 |  |
| Content-Range  RFC 2616 (HTTP/1.1) Section 14.16 | 是 | 返回文件范围，示例如下：  bytes 500-1233/1234 (范围上载)  或  bytes 0-1233/1234 (类似全部上载)  需要满足Content-Range起始点<=Range起始点 |
| Content-Disposition | 是 | 返回文件描述（文件名）  格式：  attachment; filename=${URLEncode(fileName,”UTF-8”)} |
| Content-Type | 是 | 媒体类型  固定以“multipart/”开头 |

* 1. 响应
* 状态码(Status-Code)

200：上传成功

其他：错误

注：上传实际的本地读取偏移点由客户端返回的Content-Range字段指定。Content-Range起始点<=Range起始点

# 适配器



## 适配器调度

1. **任务模板解析**
2. **适配器调度运行**
3. **适配器运行方式**
   1. 串行
   2. 并行

## 适配器加载

1. **本地适配器管理**

* 管理前置节点安装包本身打包的以及下载的适配器及依赖包
* 标注系统加载情况
* 如适配器或依赖包不存在则去交换池中心下载。

1. **公共依赖加载**

加载公共依赖包。

1. **适配器类加载**

加载适配器，先查看是否已加载；如未加载则先加载适配器依赖包，再加载适配器。

## 适配器类别

1. **连接适配器**
   1. 连接器工厂

根据导入数据源参数及指定装载的解析适配器来获取连接器。

* + 连接器接口定义：

/\*\*

\* 连接器接口,先调用init方法来初始化数据源,然后通过load方法指定要加载数据源的数据或数据集合，可传入解析器，来帮忙实现InputStream->DataRecord的转换

\* 然后通过next获取数据

\* **@author** wangm

\*/

**public** **interface** IConnectorAdapter {

/\*\*

\* 根据传入的参数初始化数据源

\* **@param** params 数据源相关参数

\*/

**public** **void** init(Map<String,String> params);

/\*\*

\* 加载预获取的数据或数据集合

\* **@param** uri 数据获取标识

\* **@param** parserAdapter 解析适配器

\* **@param** params 解析参数

\*/

**public** **void** load(String uri,IParserAdapter parserAdapter,Map<String,String> params);

/\*\*

\* 是否存在未获取的数据

\* **@return**

\*/

**public** **boolean** hasNext();

/\*\*

\* 获取下一数据对象，以输入流的方式返回

\* **@return**

\*/

**public** InputStream nextInputStream();

/\*\*

\* 获取下一数据对象，以DataRecord方式返回

\* **@return**

\*/

**public** DataRecord nextDataRecord();

/\*\*

\* 返回当前连接器代码

\* **@return**

\*/

**public** String adapterCode();

/\*\*

\* 关闭连接器相关资源

\*/

**public** **void** close();

}

* 1. 解析适配器

将连接器获取的InputStream转换成DataRecord

* + 接口定义：

/\*\*

\* 解析器

\* **@author** wangm

\*/

**public** **interface** IParserAdapter {

/\*\*

\* 加载数据流到解析器

\* **@param** inputStream

\* **@param** params

\*/

**public** **void** load(InputStream inputStream,Map<String,String> params);

/\*\*

\* 是否存在未获取的数据

\* **@return**

\*/

**public** **boolean** hasNext();

/\*\*

\* 获取下一数据对象

\* **@return**

\*/

**public** DataRecord next();

/\*\*

\* 返回当前解析器代码

\* **@return**

\*/

**public** String adapterCode();

/\*\*

\* 关闭解析器相关资源

\*/

**public** **void** close();

}

1. **转换适配器**
   1. 转换适配器

将输入的DataRecord的某字段映射到目标数据格式的DataRecord字段上，如需要计算，则收集输入值，调用转换计算适配器进行计算，计算结果填充到目标DataRecord字段上。

* + 接口定义：

/\*\*

\* 转换适配器

\* **@author** wangm

\*/

**public** **interface** IConvertAdapter {

/\*\*

\* 加载转换模板

\* **@param** template

\*/

**public** **void** load(ConvertTemplate template);

/\*\*

\* 将源DataRecord计算合并到目标DataRecord上

\* **@param** source 源数据

\* **@param** target 目标数据

\* **@param** anchor 锚点，表示将源数据合并到目标数据那个点上，用冒号连接层级字段来表示，例如：Book.Creator

\* **@return** 合并后的数据

\*/

**public** DataRecord merge(DataRecord source,DataRecord target,String anchor);

/\*\*

\* 转换适配器代码

\* **@return**

\*/

**public** String adapterCode();

}

* 1. 转换计算适配器

采用不同的计算方式将输入值计算后输出。

* + 接口定义：

1. **校验适配器**

对转换后的DataRecord进行校验。

* + 接口定义：

1. **导出适配器**
   1. 导出适配器工厂

根据导出数据源参数及指定装载的装配适配器来获取导出适配器。

* + 接口定义：
  1. 装配适配器

将DataRecord装配成OutputStream等输出

* + 接口定义：

1. **预览适配器**

面向预览库的解析、检索等服务+一套展现模板

# 系统流程

## 数据交换

### 数据交换流程



### 数据交换状态转换

## 数据处理

### 数据处理流程



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。

### 数据处理状态转换

<?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>

# 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;

}