

FLY8.1

DI应用关键技术说明

辅稳BA软件 编著

目录

[第一章 FLY-DI体系结构 3](#_Toc362412616)

[第二章 FLY-DI新特性 5](#_Toc362412617)

[2.1 术语和约束 6](#_Toc362412618)

[2.2 引擎并发机制 7](#_Toc362412619)

[2.3 全新的用户界面 7](#_Toc362412620)

[第三章 安装部署与设置 9](#_Toc362412621)

[3.1 服务器环境配置要求 9](#_Toc362412622)

[第四章 启动和登录RDI Console 11](#_Toc362412623)

[第五章 用向导开始第一个应用 12](#_Toc362412624)

[第六章 数据源管理 20](#_Toc362412625)

[6.1 新建数据源 21](#_Toc362412626)

[6.2 修改数据源 22](#_Toc362412627)

[6.3 删除数据源 22](#_Toc362412628)

[第七章 表映射关系配置 23](#_Toc362412629)

[第八章 Notification邮件通知配置 26](#_Toc362412630)

[第九章 SourceModule详解 27](#_Toc362412631)

[9.1 生产引擎 28](#_Toc362412632)

[9.2 运行期 28](#_Toc362412633)

[9.3 属性 29](#_Toc362412634)

[9.4 过滤器 31](#_Toc362412635)

[第十章 TargetModule详解 34](#_Toc362412636)

[10.1 运行期属性 35](#_Toc362412637)

[10.2 属性 36](#_Toc362412638)

[10.3 事务拦截器 37](#_Toc362412639)

[第十一章 RDI计划脚本详解 43](#_Toc362412640)

[11.1 命令简介 44](#_Toc362412641)

[11.2 上下文属性 46](#_Toc362412642)

[11.3 用脚本设计器生成脚本 46](#_Toc362412643)

[第十二章 RDI目录和系统配置文件 48](#_Toc362412644)

**背景**

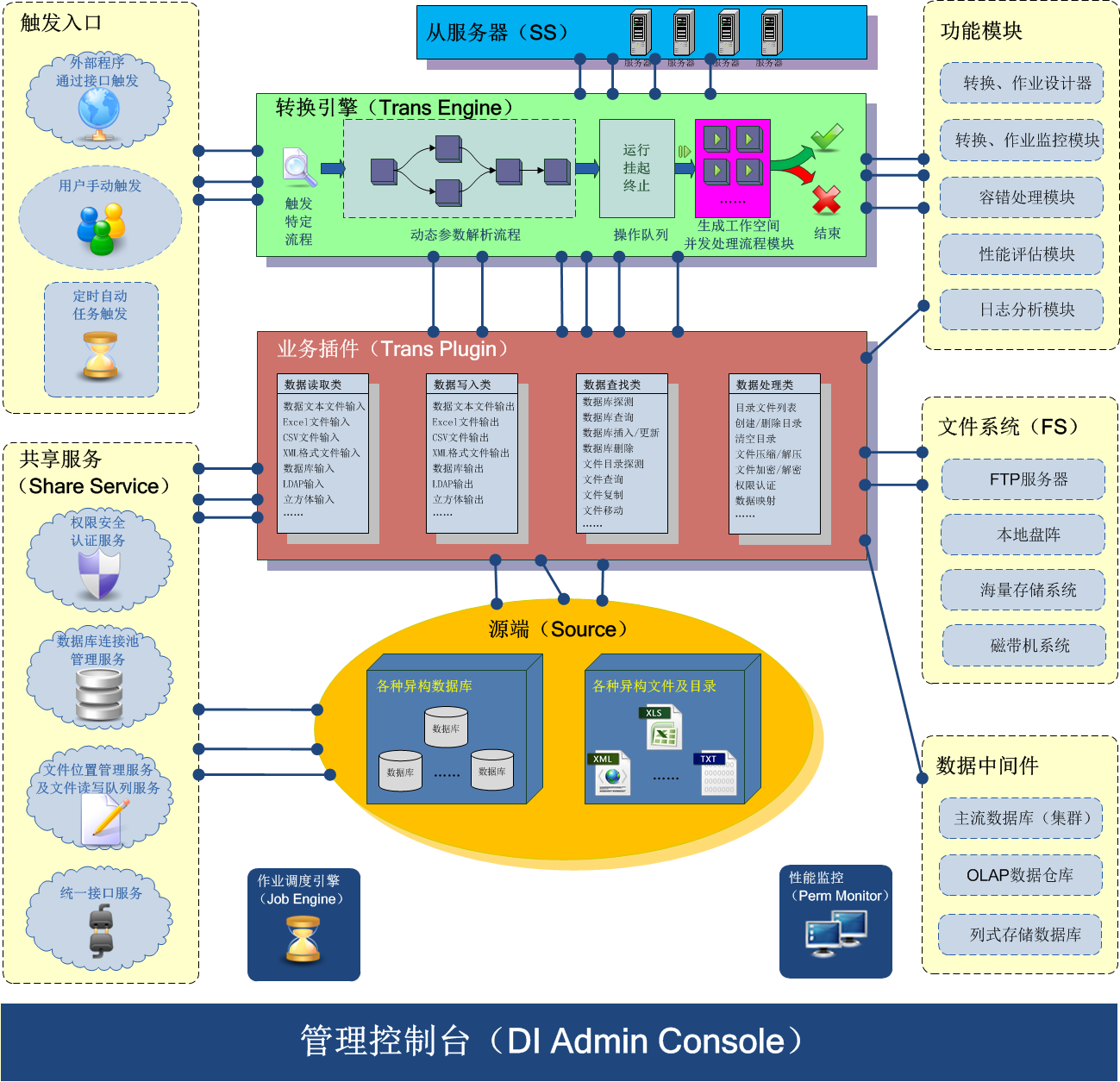
FLY8 DI是通过协同平台扩展的数据整合一体化开发运行平台，是辅稳BA（商业分析）软件在云计算环境下的重要战略产品。该产品可以帮助您轻松完成数据抽取，转换，加载，分析等的一整套数据处理流程。

其主要核心元件如下：

* FLY-TRANS用于完成具体数据的转换业务，采用复制灵活的转换插件组合完成复杂的业务逻辑；
* FLY-JOB用于进行定时作业处理，将具体的转换业务自动化批量执行；
* FLY-FS用于集中式数据文件和文档管理，支持本地盘阵存储，也支持FTP和SFTP存储，同时可进行相关索引查找；
* FLY-DW是一个行列混合存储的数据仓库，用于高效完成海量数据处理和分析

# FLY-DI体系结构

FLY DI基于分布式和并行计算技术，内部具有高缓存，高并发的架构，实现了高性能的准实时数据整合和灵活的部署模型。其工作原理简单描述如下图：



* DI Trans Engine 负责实例化转换（Trans），操作相应的插件从源端数据库、数据仓库、列存数据库、文件系统获取数据，组合各种的业务插件进行数据加工；
* DI Trans Plugin 转换插件体系，提供丰富的业务插件；
* DI Job Engine 负责进行平台作业的调度工作，自动化完成重复性业务；
* DI Share Service 提供其它平台与该平台进行整合的接口服务；
* DI Perm Monitor 负责平台和相关资源的性能监控工作；
* DI Admin Console 用来管理DI Server实例中的转换、作业和资源

# FLY-DI新特性

***本章内容概要：***

* ***术语与约定***
* ***引擎并发机制***
* ***全新的用户界面***

革新版的DI在原有架构基础上进行了深层的重构，从功能和界面都给客户耳目一新的感受。

## 术语和约束

* DI：

中文名称为数据整合，涵盖了传统的ETL的内涵和外延。ETL即Extraction-Transformation-Loading的缩写，中文名称为数据抽取、转换和加载。ETL是数据抽取（Extract）、转换（Transform）、清洗（Cleansing）、装载（Load）的过程。DI平台负责将分布的、异构数据源中的数据如关系数据、平面数据文件等抽取到临时中间层后进行清洗、转换、集成，最后加载到数据仓库或数据集市中，成为联机分析处理、数据挖掘的基础。

* 资源库：

资源库用来保存连接信息和转换信息。用户通过图形界面创建的的数据库连接和数据转换任务可以保存在资源库中，这意味着为了从数据库资源中加载一个转换，你必须连接相应的资源库。资源库可以是各种常见的数据库。

* 业务数据库：

存储各种业务数据的数据库，DI平台可以将不同业务数据库中的数据作为源数据，通过抽取、转换、清洗和装载导入到同一数据仓库或者采用其他形式进行存放中。

* 数据仓库：

数据仓库，英文名称为Data Warehouse，可简写为DW。数据仓库是一个面向主题的、集成的、不可更新的、随时间不断变化的数据集合，它用于支持企业或组织的决策分析处理。

* 列式存储数据库：

列式数据库是以列相关存储架构进行数据存储的数据库，主要适合与批量数据处理和即席查询。用于大数据环境下的分析应用。

* 多维分析（OLAP）：

联机分析处理（On-Line Analytical Processing,简称OLAP），是一套以多维度方式分析数据，而能弹性地提供积存(Roll-up)、下钻(Drill-down)、和枢纽分析(pivot)等操作，呈现集成性决策信息的方法，多用于决策支持系统、商务智能或数据仓库。其主要的功能，在于方便大规模数据分析及统计计算，对决策提供参考和支持。与之相区别的是联机交易处理(OLTP)。

1）OLAP需以大量历史数据为基础配合上时间点的差异并对多维度及汇整型的信息进行复杂的分析。

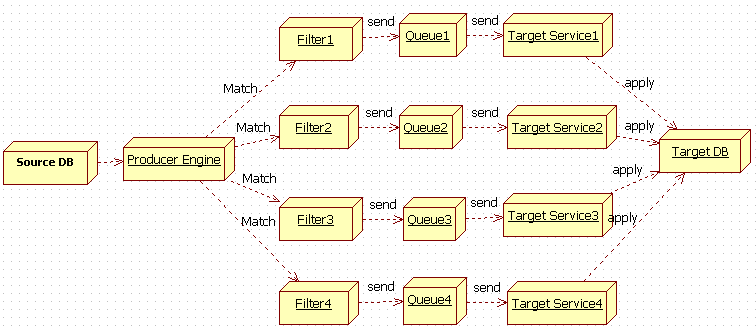
2）OLAP需要用户有主观的信息需求定义，因此系统效率较佳。

OLAP的概念，在实际应用中用广义和狭义两种不同的理解。广义上的理解与字面意思相同，即针对于OLTP而言，泛指一切不对数据进行输入等事务性处理，而基于已有数据进行分析的方法。但更多的情况下OLAP是被理解为其狭义上的含义，即与多维分析相关，基于立方体(CUBE)计算而进行的分析。

## 引擎并发机制

这个特性主要是基于这样一个事实，不同插件节点的分析事务速度有所不同，特别是当目的端数据库是列式存储数据库的情况下。于是我们把转换架构引擎默认调整为生产者消费者多线程结构，同时也可以将分析得到的事务根据模块分发到不同的从服务器服务队列，每个服务队列对应一个step service。这样减少对源库的压力的同时，目的端数据库的并发应用事务的能力也得以发挥，可以很大的提高性能。

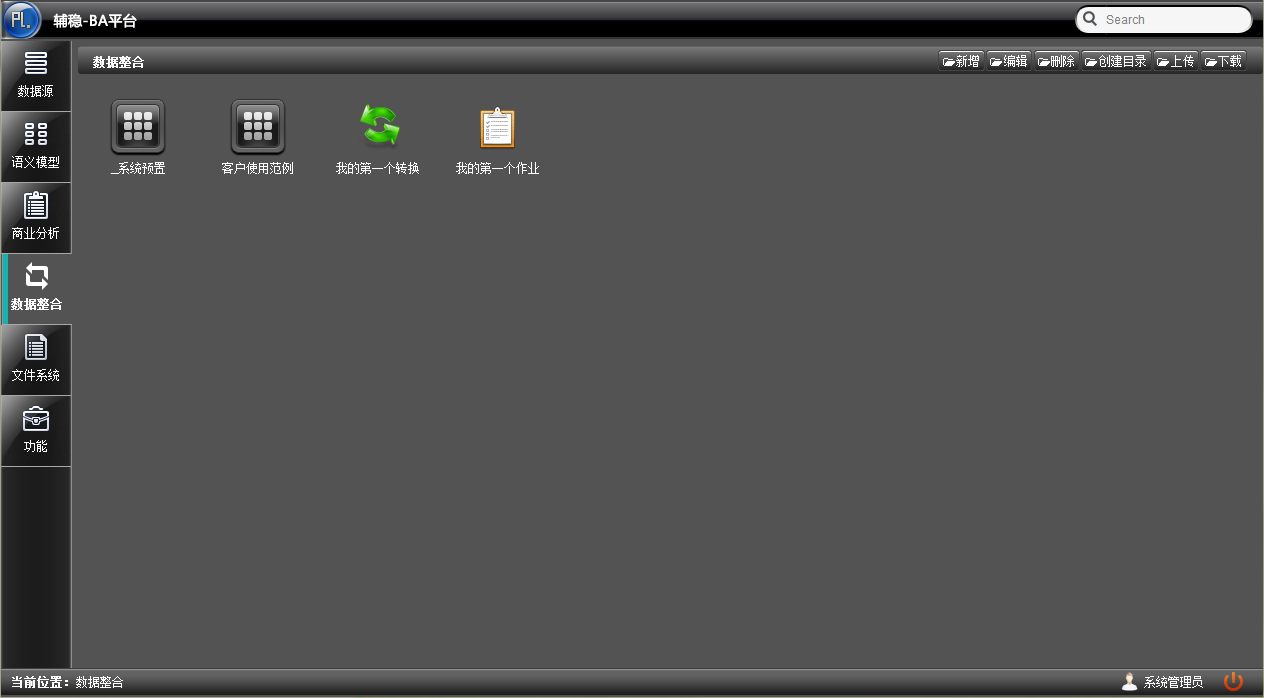
其模型如下图：



## 全新的用户界面

新的用户界面采用Windows浏览器风格，同时有向导功能，界面更加美观的同时用户操作将更加简便灵活。

* **Windows文件管理器风格**



* **可视化编辑器**

在以下的部分中我们将结合用户界面详细讲解DI的各项功能。

# 安装部署与设置

## 

## 3.1 服务器环境配置要求

1. 操作系统平台

|  |  |  |  |
| --- | --- | --- | --- |
| 应用服务器操作系统 | 版本（补丁） | 中间件类型 | JDK版本 |
| Linux-RedHat(x64，多核) | Enterprise AS5 update 4 | Websphere 7.0.0.21 Weblogic11  Tomcat 6.0.16  Jboss 5.1.0 | Java SDK 6u24  IBM SDK,Version 6.1  jrockit 1.6 |
| Linux – SUSE(x64，多核) | SUSE 10 sp2 | Websphere 7.0.0.21 Weblogic11  Tomcat 6.0.16  Jboss 5.1.0 | Java SDK 6u24  IBM SDK,Version 6.1  jrockit 1.6 |
| IBM AIX | 6.1(6100-00-04) | Websphere 7.0.0.21 | IBM SDK Version 6.1 |
| Windows 2008 R2(x64，多核) | Windows 2008 R2 Enterprise x64 R2 | Websphere 7.0.0.21 Weblogic11  Tomcat 6.0.16  Jboss 5.1.0 | Java SDK 6u24  IBM SDK,Version 6.1  jrockit 1.6 |
| Solaris Sparc | Version 10 with Patch Cluster dated 1/07/08 or later(64 bit kernel support) | Websphere 7.0.0.21 | IBM SDK for Solaris,Version6.1 |
| Solaris x64 | Version 10 with Patch Cluster dated 1/07/08 or later(64 bit kernel support) | Websphere 7.0.0.21 | IBM SDK for Solaris,Version6.1 |
| |  |  |  |  | | --- | --- | --- | --- | | HP-UX IA64 |  |  |  | | 11.iv3 Patch Bundle dated Sep 2007(64bit kernel support) | Websphere 7.0.0.21 | HP JDK for J2SE HP-UX 11i platform,adapted by IBM for IBM Soft,Version 6.0 |
|  |  |  |  |

注意：

没有明确推荐的安装平台不建议使用，用户坚持使用未推荐平台可能会产生的不可预测问题和某些功能的表现差异。

使用虚拟机作为应用服务器时，VMWARE ESXI 平台最低版本为5.0 U1；redhat linux平台的kvm最低版本为2.6.32。

1. 应用规模硬件配置推荐

简明硬件建议表，本表所写关键硬件要求以Intel Xeon 3.0G为标准计算：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 并发转换数量 | **1～20** | | **20～50** | | **50～100** | | **>100** | |
| 配置 | 应用 服务器 | 数据 服务器 | 应用 服务器 | 数据 服务器 | 应用 服务器 | 数据 服务器 | 应用 服务器 | 数据 服务器 |
| CPU数量(物理核) | 8 | 8 | 8 | 16 | 8×2 | 16×2 | (1) | (1) |
| 内存配置(G) | 8 | 16 | 16 | 32 | 16x2 | 32×2 | (1) | (1) |
| 磁盘数量(块) | 1 | 磁盘阵列 | 1 | 磁盘阵列 | 1 | 磁盘阵列 | (1) | (1) |

注意：

1. 应用服务建议使用PC Server，数据库服务器建议使用小型机，硬盘使用SCSI接口10000转以上磁盘或磁盘阵列。
2. 表中的CPU数量指的是物理核心数，8核对应为2Cx4core。
3. 此表的统计方法是按照一般复杂度（节点数<=10）的转换计算，作业的数量同时也会对并发数量有些影响。
4. 高并发或者多用户实时请求转换的应用场景，请联系本公司技术支持部门协商评估应用集群方案和硬件及网络方案。
5. 数据库支持情况

* 使用RDI的关闭命令：**>rdi stop** 。
* 在Windows或Linux命令行用Ctrl + c键退出。

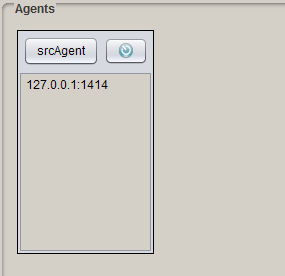
# 启动和登录RDI Console

用户可以在<RDI\_HOME>/bin下执行rdi console打RDI管理控制台；RDI管理控制台也可以单独运行，拷贝<RDI\_HOME>/console到任意目录，运行console.bat脚本。

在顶端的输入框中输入服务器端的IP地址和端口号（其输入格式为 : <IP>:<port>）,回车。如图所示：



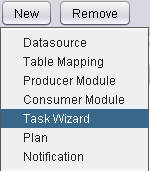
Enterprise Console会自动记录用户浏览历史，下次登录Console或者点击 按钮打开新登录页面的时候会出现以前浏览过的节点快照，可以点击”srcAgent”按钮进入。如图所示：



# 用向导开始第一个应用

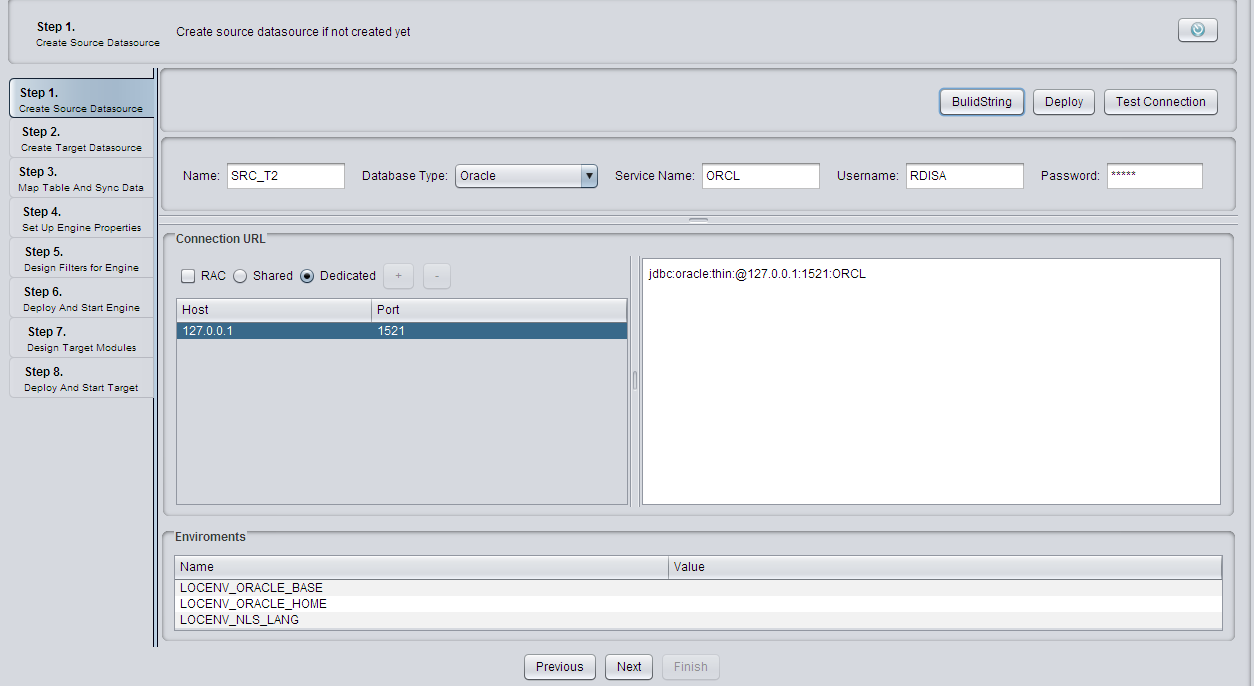
用户按照最后一章“系统初始化安装及数据库配置”中介绍的步骤进行系统安装以后，我们可以用任务向导来开始我们第一个应用。

登录服务器后，点击”New”按钮，选择”Task Wizard”选项打开任务向导。



1. **步骤一：添加Source Module数据源。**

各个信息如图所示：



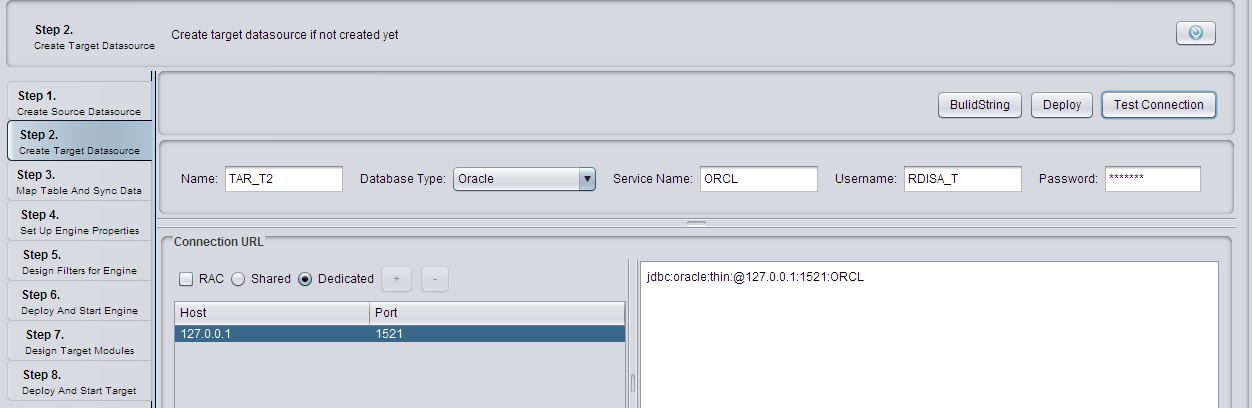
其中:name 表示源端数据源名 ；database Type 表示使用的数据库类型；service Name 使用的数据库实例（数据库名）；后两项是用户名和密码。

Host和Port是数据库所在的主机地址和数据库对应的端口号，输入完端口号后要回车，这样才表示输入完毕，否则按buildString无反应。

用户在界面上填写好各项信息，然后点击在文本域中会显示数据库连接字串，可以点击按钮测试配置是否成功，出现“connection work fine”后表示测试成功，可以点击按钮部署这个数据源。除了采用这种添加方式之外，还可以使用”New→Datasource”方式来配置数据源。

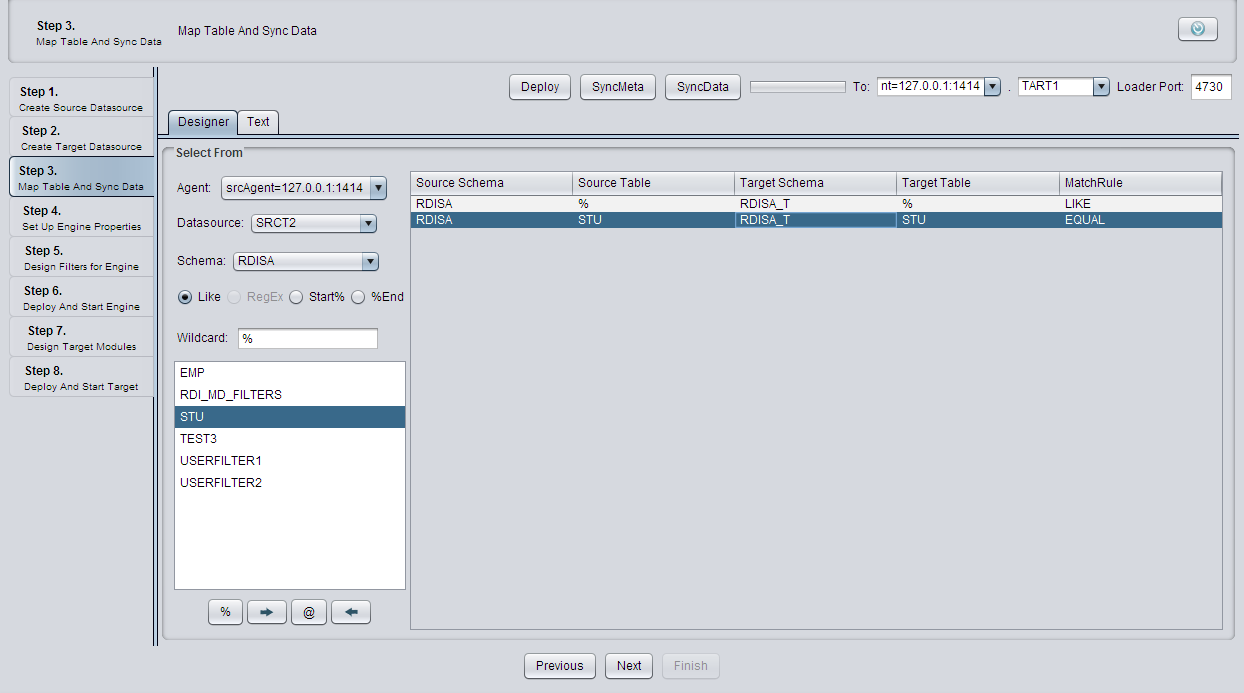
如果在使用向导之前已经配置好了所需的数据源，用户就可以点击向导底部的跳过这一步。

1. **步骤二：创建TargetModule数据源**



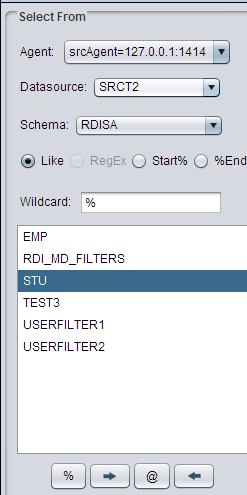
这里是配置目的端数据源的信息，和源端数据源类似。如果目的端数据源以前配置好，也可以跳过该步。

1. **步骤三：表映射关系配置**



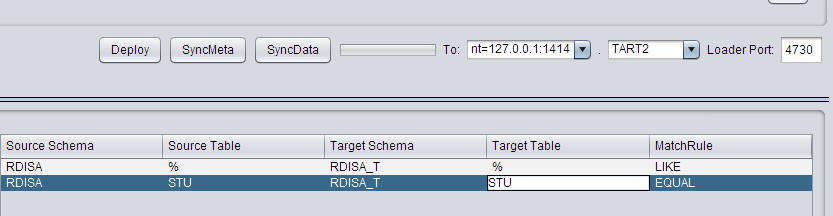
此项用于配置源端数据库表和目的端数据库表的对应关系，用户可以用此关系在目的端创建相应的表以及从源表向其中导入数据。目前只支持目的端是GBase的数据库。

用户通过设计器面板选择要镜像的表：



参数：agent 默认即可；Datasource 数据源：第一步配置的源端数据源，Schema：表的模式，Oracle一般模式名就是用户名，GBase一般模式名是数据库名； 表示匹配模式；

下面的是数据源的所有表，可以通过添加和删除表，表示源端数据库向目的端数据库要同步的表（表结构和表数据）。点击选入，用户也可以用LIKE语法匹配（Start%，%End也属于LIKE匹配），然后点击选入，也可以点击从文件中导入。点击可将选入的删除。在右边表映射的表格中可以修改目的schema和表名，如图：



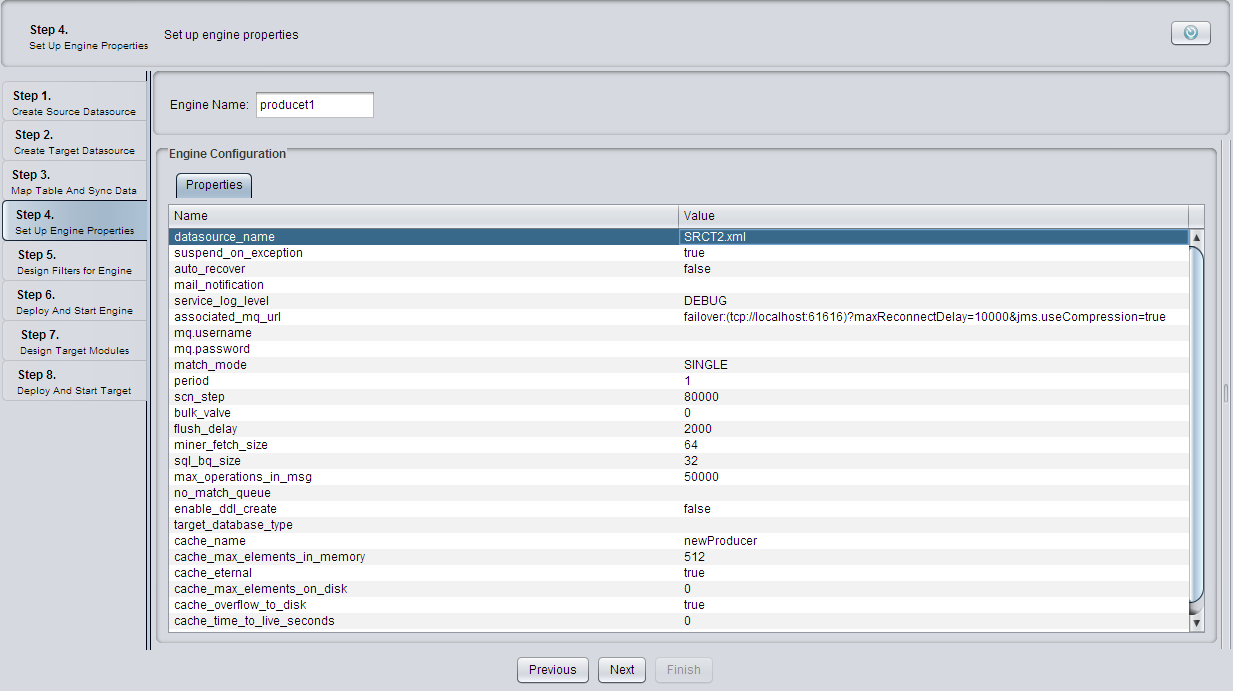
这里Source Schema和Target Schema 要注意的是不同数据库是不同的，Oracle一般模式名就是用户名，GBase模式名要用数据库名；而且Oracle数据库一定要注意大小写，如果用户名是大写，那么schema名一定大写。

第一条表示可以把源端RDISA用户的所有表及数据都导入RDISA\_T数据库中。RDISA\_T的用户在设置目的端数据库时已经指定。第二条表示可以把源端RDISA用户的表及数据STU导入RDISA\_T数据库中。

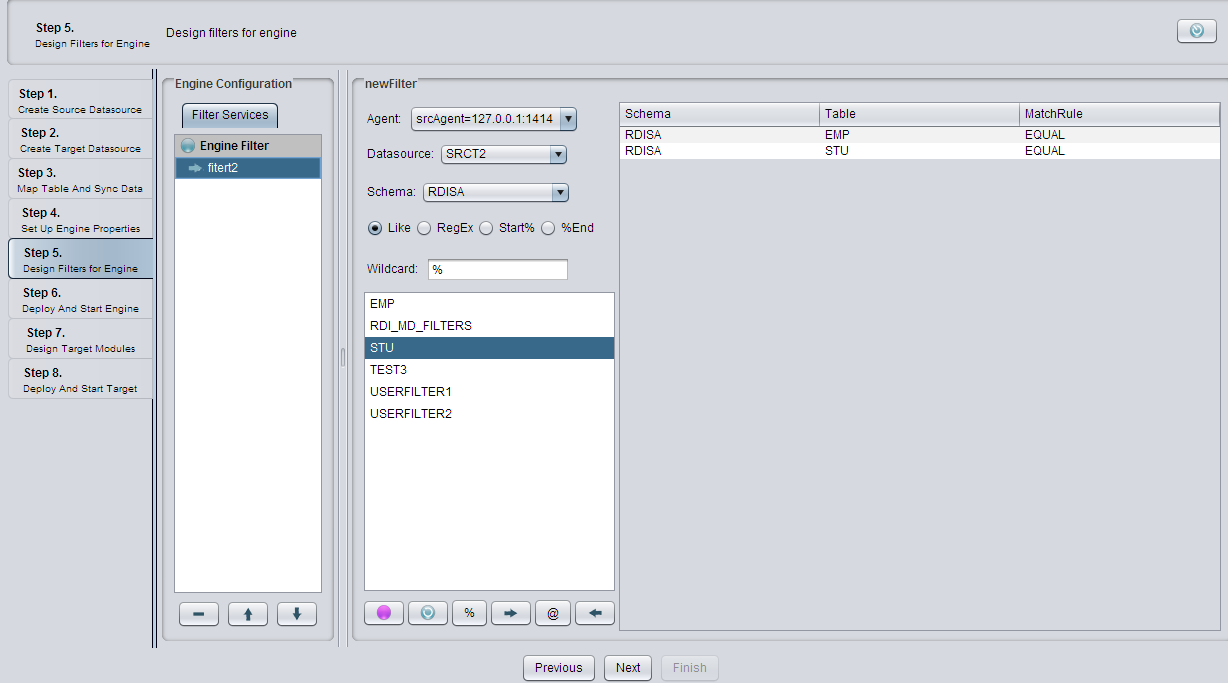
其中导入表结构用，导入表数据用。

1. **步骤四：创建Producer Engine并设置其属性**

在这一步中，我们需要填写Engine名称，然后要填写datasource\_name属性为我们在第一步配置的数据源，这个属性是必须的，其他属性用默认值即可，我们将在后面的章节中介绍其他属性的意义。这样ProducerEngine就连接到这个数据源，如图所示：



1. **步骤五：为生产引擎设计过滤器**



在这一步中需要设计两类过滤器：

第一类是引擎过滤器Engine Filter，Engine Filter是第一级过滤器，该过滤器可以使用LIKE,EQUAL规则进行匹配，但不支持正则表达式匹配。一个引擎只有一个Engine Filter。

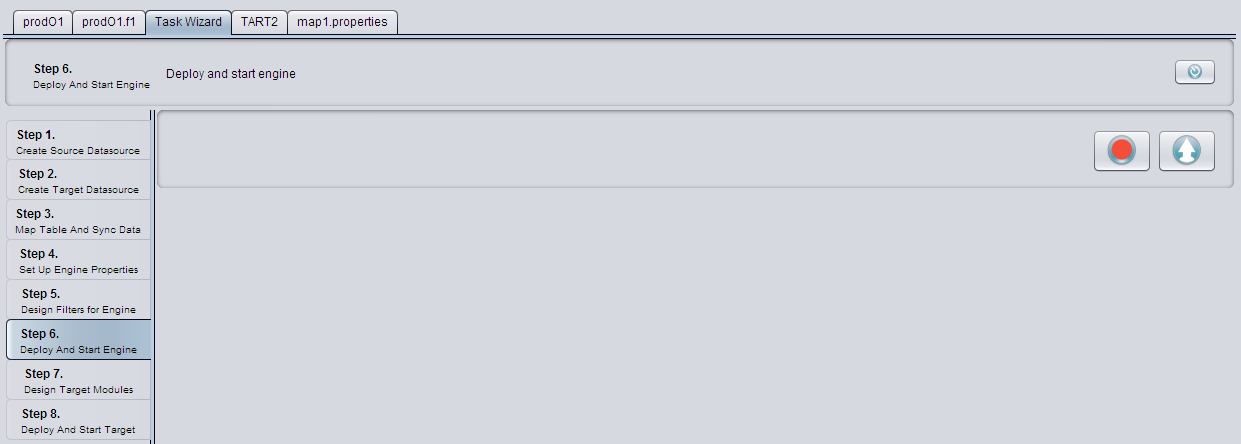
第二类是自定义的二级过滤器，该过滤器可以使用LIKE,EQUAL和正则表达式规则进行匹配，用户可以顺序添加多个二级过滤器。当前默认的过滤器为Engine Filter,如面板边框显示,点击按钮保存当前过滤器，点击新建一个过滤器，然后选择要匹配的表，选择好以后要点保存新建的二级过滤器，否则不保存，在右端是看不到新建的二级过滤器的。我们选择一个表，点击加入右边的表格中（确保在目的端的数据库scheme下也已经创建了对应的表，对于表映射关系配置成功的源端和目的端数据库，要先同步表结构和表数据）。

这里要注意的是RDI\_MD\_FILTERS不能添加到右边的表格中。

生产引擎会先用Engine Filter定义的规则从源端数据库分析得到事务，然后依次根据用户定义二级过滤器进行匹配（如果匹配规则有重复，那么后匹配到的将被忽略），并将匹配到的事务发送到相应的服务队列。

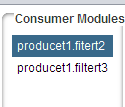
1. **步骤六：部署并启动配置好的生产引擎**

点击部署引擎，点击启动引擎，如图所示：

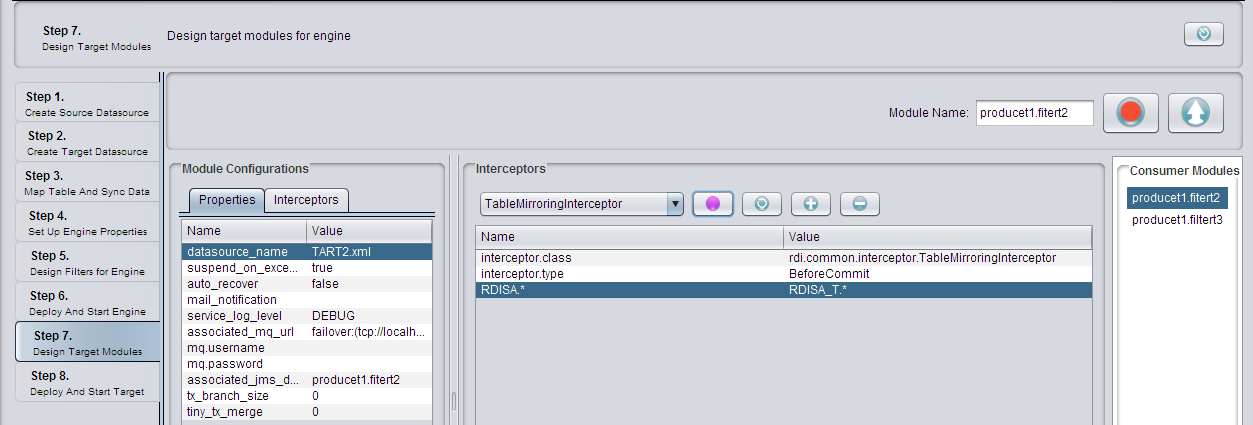


1. **步骤七：生成和配置相应的TargetModule**

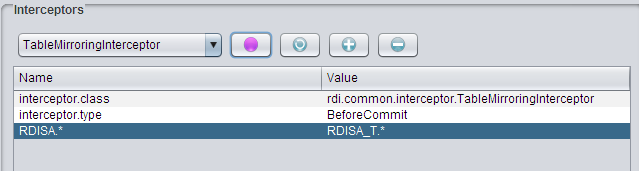
生成的TargetModule与Producer Engine的自定义过滤器是一一对应的，如右边的target module所示：



依次双击选择每个TargetModule并为其设置属性及拦截器。如图



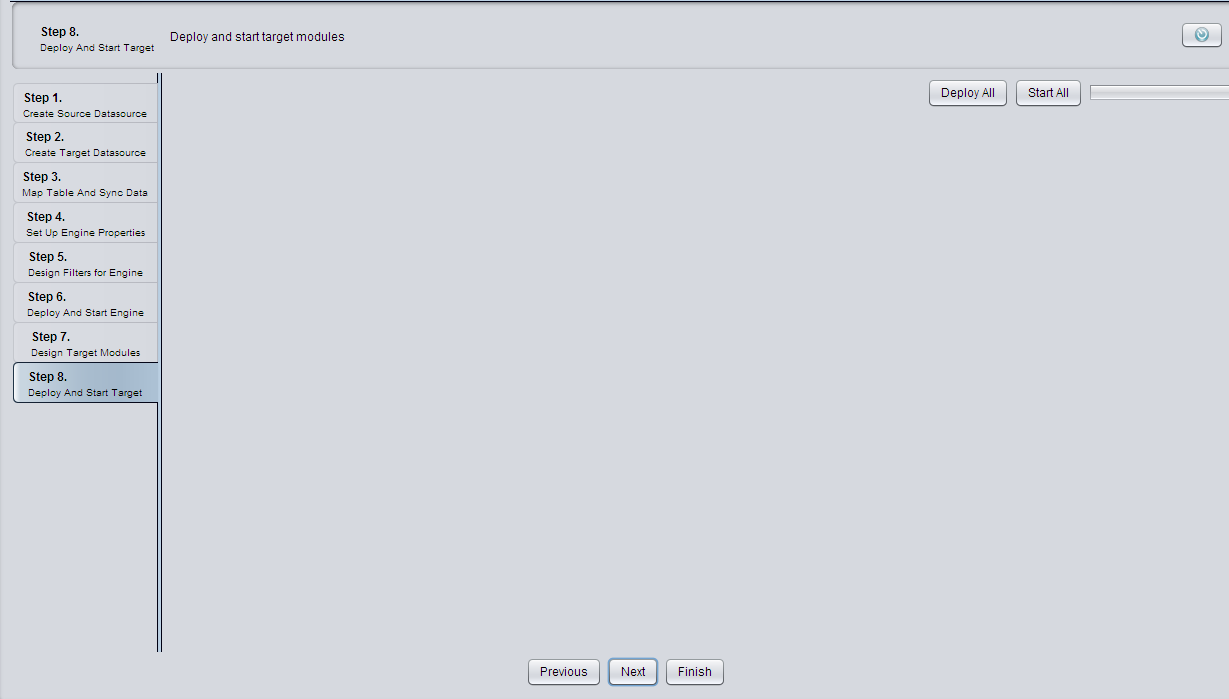
设置datasource\_name属性为目的端数据源。其他属性我们将会在下面的章节中介绍。然后为每个module添加一个TableMirroringInterceptor：



最下面一行意味着将源端Schema为RDISA的表映射为目的端Schema为RDISA\_T的表，表名不变,点击保存。这里要注意有两点：Schema的大小写；目的端Schema注意区分数据库，前面已经说明。然后点击部署该引擎，点击启动该引擎，也可在下一步全部部署和启动。

1. **步骤八：部署并启动所有的TargetModule**

点击部署所有的target module，点击启动所有的target module并以进度条显示进度。当所有操作完成后，单击按钮可结束向导任务。如果在向导配置过程中，出现错误可以点击按钮返回进行修改。其界面如图所示：



当全部部署完并启动后要检查producer services 和 comsumer services 是否都正常启动，如果有哪个没有启动，点击启动。

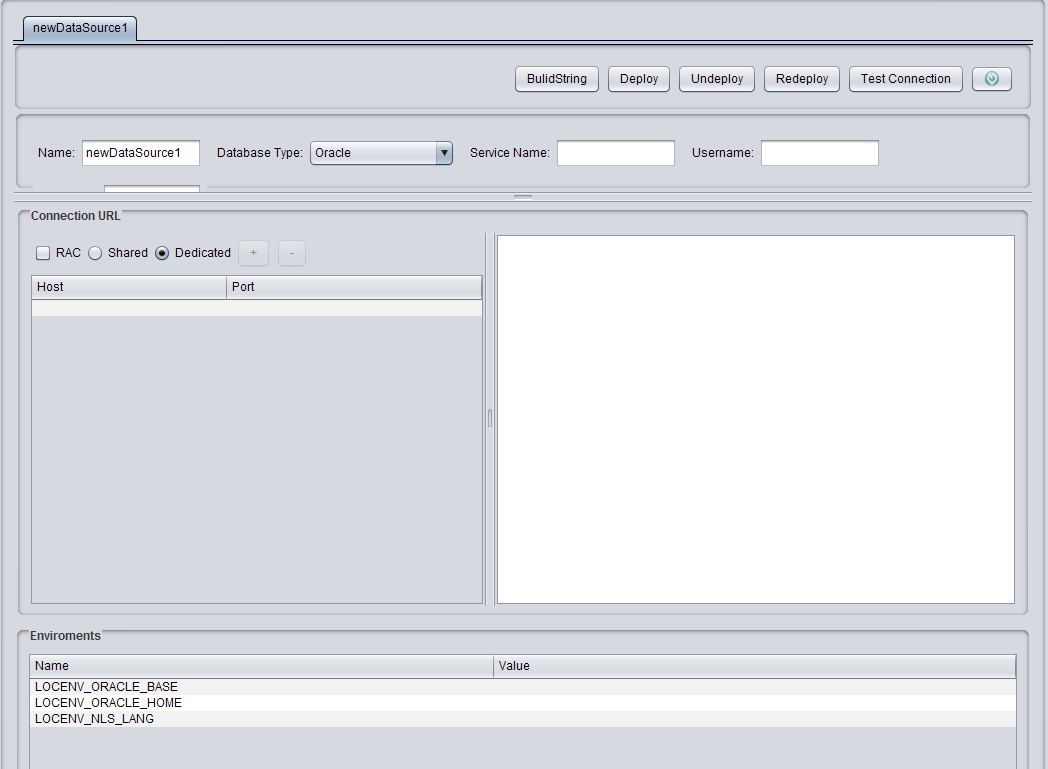
# 数据源管理

***本章内容概要：***

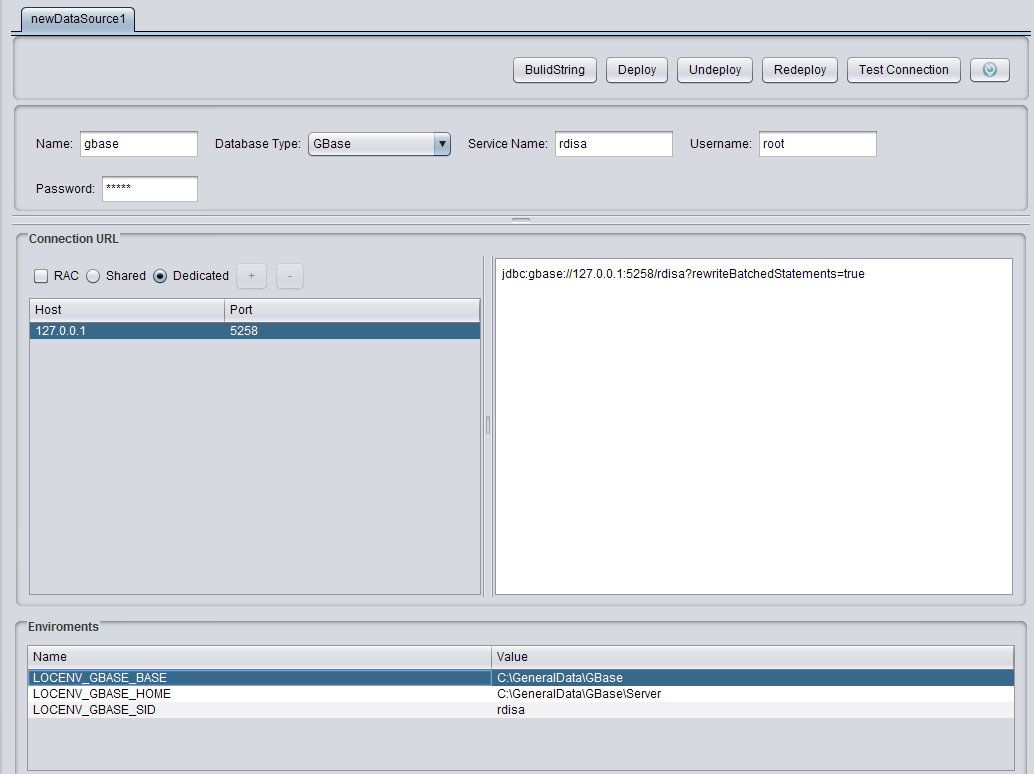
* ***新建数据源***
* ***修改数据源***
* ***删除数据源***

## 新建数据源

打开注册的服务器界面，点击 ”New” 按钮，并选择 ”DataSource” ，即可打开数据源的配置界面，如下图所示：



1. 在界面中输入数据源的配置信息后，点击  按钮，生成RDI连接数据库的语句，然后点击  按钮，将该数据源配置到服务器中。
2. 对于已经编辑好的数据源，可以点击按钮，以测试该数据源的配置是否正确。
3. 如果数据库是Oracle RAC集群，则选中RAC后可以添加多个IP地址和端口号。



1. 当部署数据源成功后，新建的数据源会自动添加到左侧的System Properties列表中.通过点击其中的列表项查看已配置的数据源。

## 修改数据源

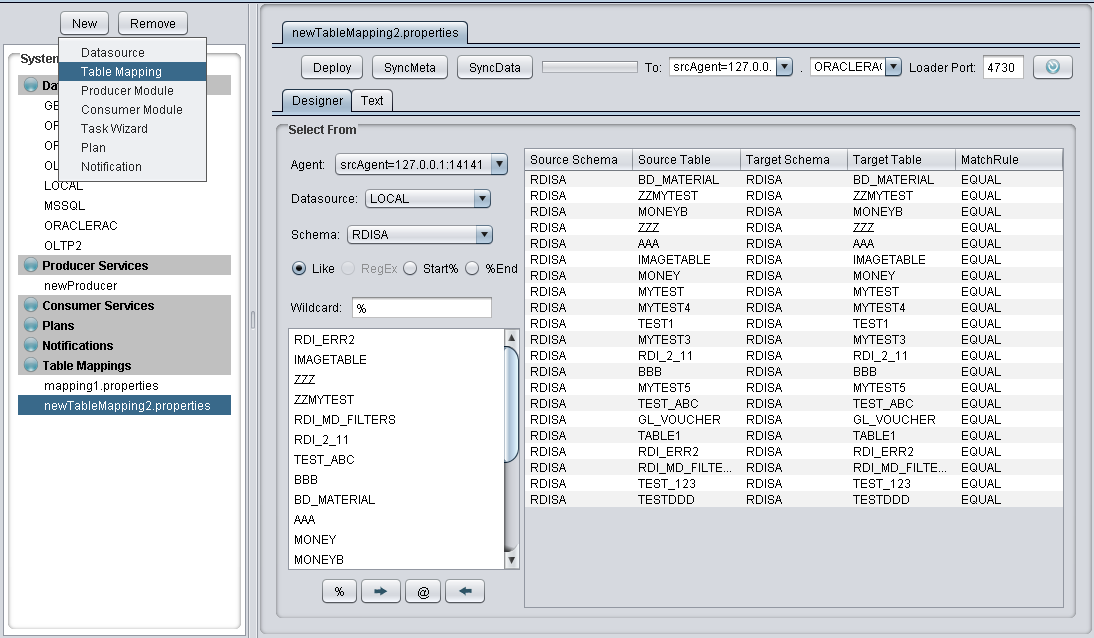
在左侧的系统属性列表中，选中需要修改的数据源并点击打开配置界面，可对该数据源中的内容做出修改。修改成功后，需重新点击”BuildString” 按钮，生成相应的连接数据库语句后，再点击“Deploy” 按钮，将该数据源配置到服务器中。

## 删除数据源

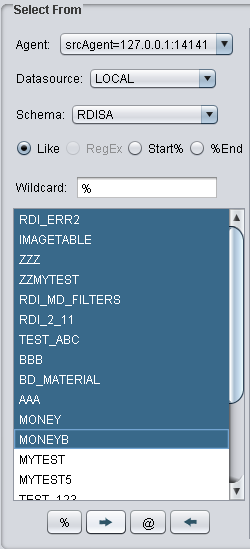
如果配置的数据源已经失效，无法与数据库建立连接，或者该数据源不再需要时，选中左侧的系统属性列表中的数据源，点击列表框上面的将其删除，或者双击在打开的界面中点击”Undeploy”按钮，即可删除此数据源。

# 表映射关系配置

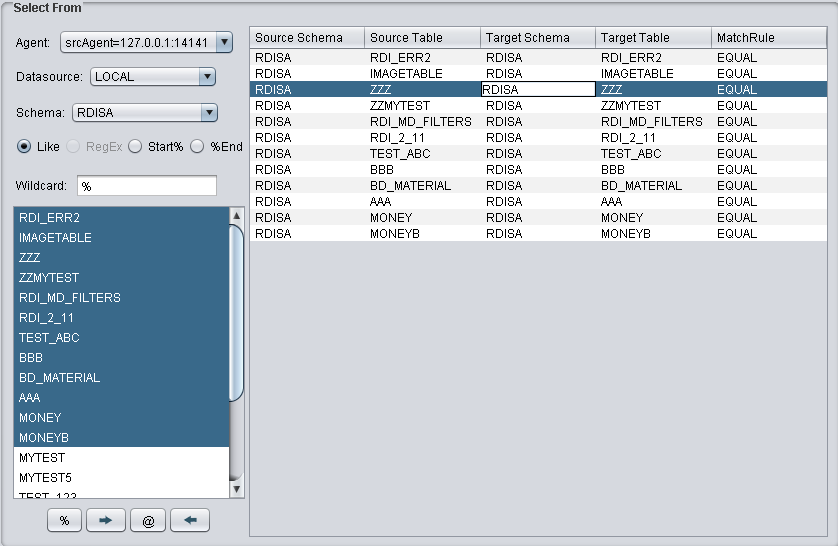
此项用于配置源端数据库表和目的端数据库表的对应关系，用户可以用此关系在目的端创建相应的表以及从源表向其中导入原始数据。如图所示：



用户通过设计器面板选择要镜像的表：



点击选入，用户也可以用LIKE语法匹配（Start%，%End也属于LIKE匹配），然后点击选入，也可以点击从文件中导入。点击可将选入的删除。在右边表映射的表格中可以修改目的schema和表名，如图：



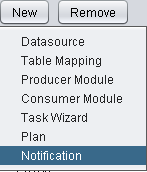
点击可以将此映射部署，以便在计划任务中使用，点击在目的端数据库创建相应的表，点击向目的数据库导入相应数据。其中



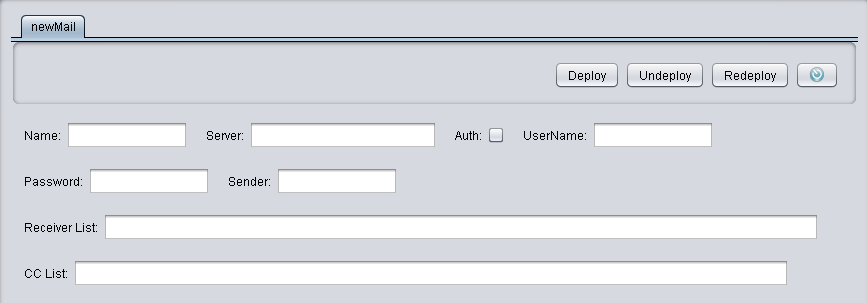
分别表示目的服务器，目的数据源和Ora28a Agent的端口号（导入数据的时候需要）。

# Notification邮件通知配置

RDI Notifacation用于RDI系统在系统发生异常等情况下给用户发送通知信息，目前主要用电子邮件的方式，从New按钮菜单选择Notififation如图配置：



进入配置界面，多个接受者可以用逗号分隔，如图：



# SourceModule详解

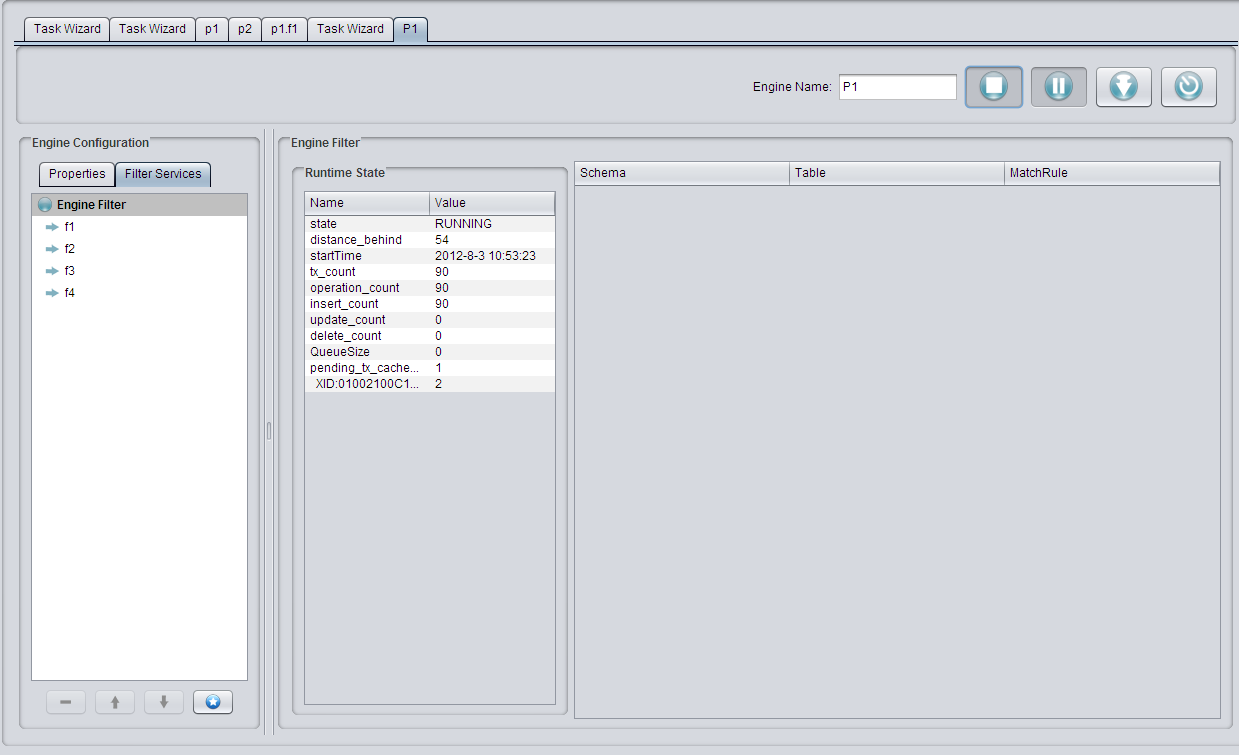
***本章内容概要：***

* ***生产引擎***
* ***运行期***
* ***属性***
* ***过滤器***

SourceModule是源端服务的配置描述，现在包含一个生产引擎及其配置信息。

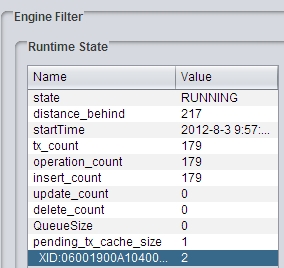
### 生产引擎

生产引擎负责按照其属性配置从源端数据源分析日志，然后将日志按照过滤器进行匹配然后发送到服务队列。生产引擎部署后就成为一个SourceService。用户可以通过RDI Console开启/关闭、中断/恢复，以及监控Source Service的运行状态，如图：



### 运行期

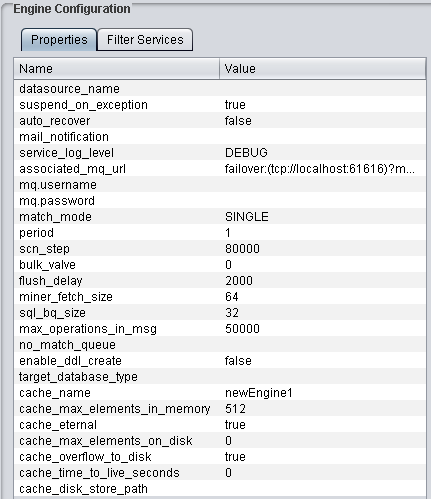
源服务运行期状态属性有下图所示：



* state表示运行状态，有STOPPED(已停止),RUNNING(正在运行),SUSPENDED(暂停)
* startTime状态项，用来记录源服务启动的时间。因为事务和操作计数是从服务启动开始累加的。校对源和目的的事务和操作数是否一致需要源和目的服务是同一次启动的才有意义。此项就是用于确定源和目的服务是否同时启动
* distance\_behind表示源服务现在分析进度与当前数据库执行落后多少个操作，distance\_behide大， 说明源服务事务分析慢，和目的服务的QueueSize配合来查看系统的性能瓶颈
* tx\_count表示从源服务启动到现在已经分析了多少个事务
* operation\_count表示从源服务启动到现在已经分析了多少个操作
* insert\_count表示从源服务启动到现在已经分析了多少个insert操作
* update\_count表示从源服务启动到现在已经分析了多少个update操作
* delete\_count表示从源服务启动到现在已经分析了多少个delete操作
* QueueSize用于目的服务
* Pending\_tx\_cache\_size 事务分析二级缓存中尚未提交的事务数量
* XID:<transaction ID> 该未提交的事务中目前的操作数

### 属性

属性列表中列出了生产引擎运行所需要的属性，其中一些是必要属性，另外一些是性能调优的参数，如图：

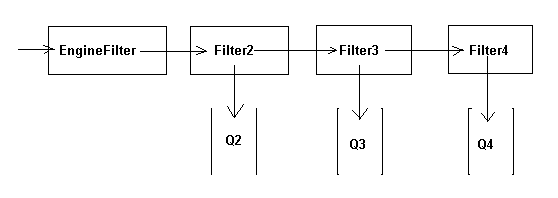


详细说明如下：

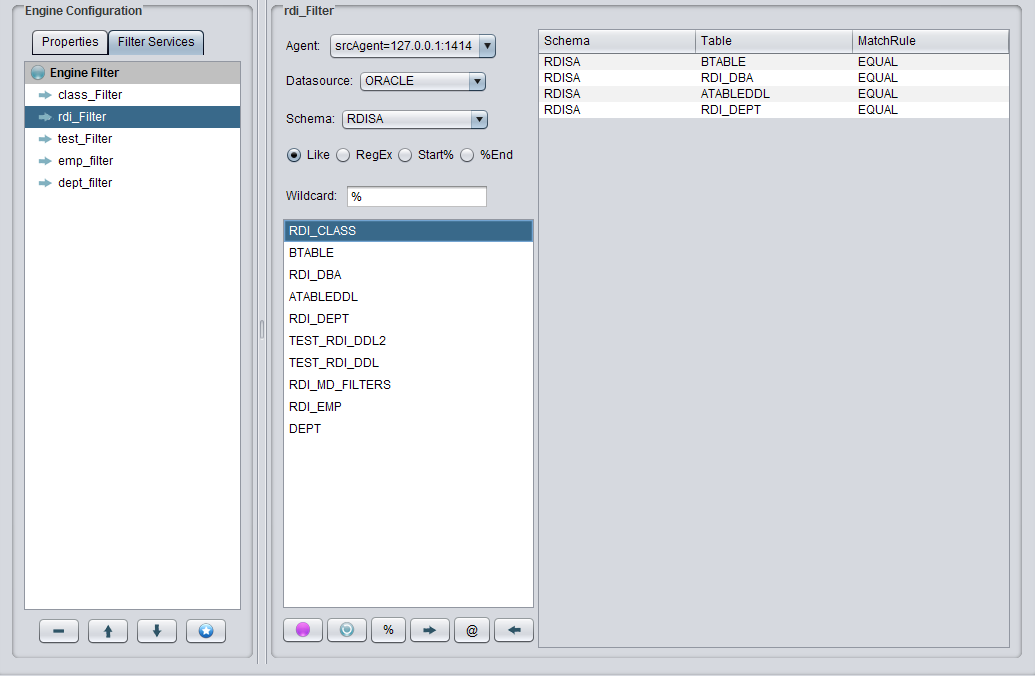
|  |  |
| --- | --- |
| Datasource\_name | 生产引擎中使用的数据源名 |
| suspend\_on\_exception | 当生产引擎执行遇到异常时是否suspend该服务 |
| auto\_recover | 当遇到Message Queue或者数据库连接中断时，是否自动恢复,当生产引擎配置为suspend\_on\_exception=true,并且service因为异常而suspend时，此项无效，不自动恢复。 |
| mail\_notification | Mail通知配置 |
| Service\_log\_level | 服务日志的输出级别 |
| mq.url | 用于连接消息中间件的URL.目前默认提供activeMQ作为消息中间件。用户可以多种方式连接到消息中间件。例如：  server.home/conf/agent.properties有如下配置：  mq.url=vm:(broker:(tcp://localhost:8080)?persistent=true)?marshal=false  表明Source Server 用本地VM协议连接到activeMQ，并且他们运行在同一个进程中，同时activeMQ对其他客户端开放tcp协议访问。此时Target Server可使用tcp协议连接到activeMQ, server.home/conf/agent.properties 应作如下配置：  mq.url=tcp://localhost:8080  在linux环境网络环境中应该使用真实IP地址，不要使用127.0.0.1  activeMQ亦可运行于单独的进程中，以及支持http等多种协议。详细了解可见 http://activemq.apache.org/ |
| mq.username | 连接到消息中间件的用户名 |
| mq.password | 连接到消息中间件的密码 |
| scn\_step | 每一轮日志挖掘的步长 |
| period | 每一轮日志挖掘的时间间隔 (毫妙) |
| Match\_mode | 目前只只是SINGLE |
| bulk\_valve | bulk\_valve=0表示不使用事务缓存区，分析得到一个事务及时发送，对于Source端业务压力较小的情况下可以获得最好的实时性  bulk\_valve >0 用户业务压力较大的系统，Source Server使用批量缓存策略，当事务缓存区的事务的SQL操作总数超过此值时，事务缓存区中的事务将作为一批发送。 |
| flush\_delay | bulk\_valve>0 的情况下，如果事务缓存区有时务，且与上次事务发送的时间间隔大于此值，那么事务缓存区中的事务将被发送。 |
| Miner\_fetch\_size | 生产引擎的 fetch size，默认为64 |
| Sql\_bq\_size | 生产引擎二级缓存大小，默认为32，单位是K |
| enable\_ddl\_create | 表示是否支持 create table的DDL |
| Cache\_name | 缓存的名字 |
| cache\_max\_elements\_in\_memory | 缓存最大元素数目 |
| cache\_eternal | 是否用不过期 |
| cache\_max\_elements\_on\_disk | 缓存本地存储最大元素数目 |
| cache\_overflow\_to\_disk | 是否从内存溢出到硬盘 |
| cache\_time\_to\_live\_seconds | 缓存生命周期 |
| cache\_disk\_store\_path | 本地缓存存储路径 |

### 过滤器

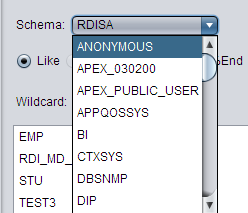
过滤器将生产引擎分析所得到得事务按照表名规则进行匹配过滤，一个生产引擎可以设置多个过滤器，并且过滤器按照顺序进行过滤。名字为EngineFilter的过滤器是生产引擎的第一个过滤器，用户必须进行设置。用户还必须定义一个或以上的自定义的二级过滤器。引擎会用EngineFilter定义的规则去分析事物，然后将所得的事务按照二级过滤器定义的顺序依次进行匹配，并发送到与之对应的服务队列，如果一个事务可以匹配到多个过滤器的规则，那么后面的过滤器将被忽略。EngineFilter支持EQUAL，LIKE（包含BeginWith和EndWith）语法规则，二级过滤器还支持正则表达式规则。工作原理如下：



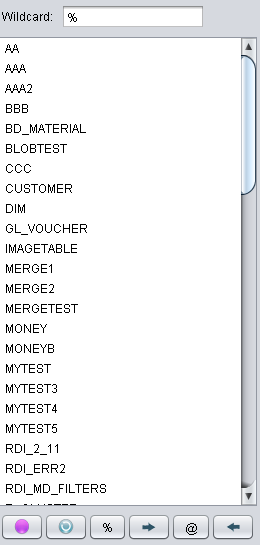
配置界面如图：



这里对于Schema中的列表时按照字母顺序从小到大进行排序的，另外如果想知道schema的名字，那么可以在点击schema的列表框时，按schema的首字母，能够更快定位。如图，可以直接按“R”就可以。



对于上面选定的schema的所有表，会显示在下图中，并且它们也是按照字母顺序排序的。

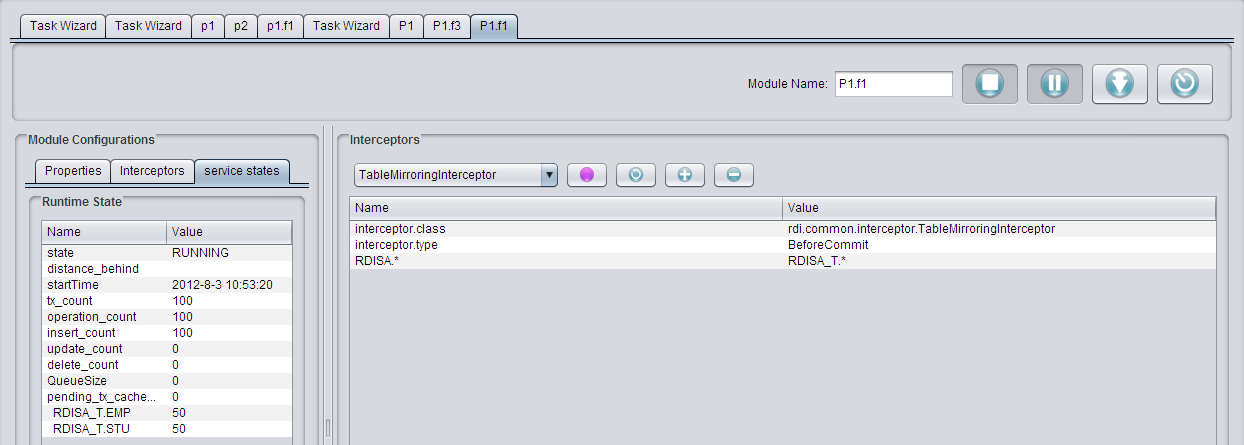


# TargetModule详解

***本章内容概要：***

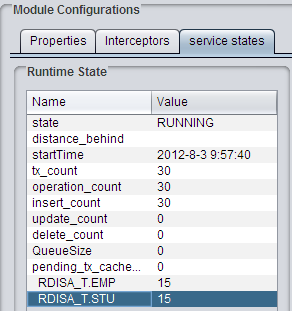
* ***生产引擎***
* ***运行期***
* ***属性***
* ***过滤器***

TargetModule从消息中间件服务队列接收事务并且应用到目的数据库。TargetModule与工作队列一一绑定，所以也可以认为和source module的二级过滤器一一对应。TargetModule部署并启动后就是一个target service，它是独立运行的线程。用户可以通过RDI Console启动/停止，暂停/恢复，部署/解部署这个服务，还可以监视运行期的状态属性，与source service相似，Distanc\_behide对target service无意义。如图所示：



### 运行期属性

目的服务运行期状态属性由下图所示：



目的服务的状态属性和源服务很相似，这里只说明有区别的地方：

* distance\_behind 在目的端无意义；
* QueueSize 表示队列中存在的消息个数，如果值大，说明目的数据库应用事务慢；和源服务distance\_behind一起来查看系统的性能瓶颈；（目的服务QueueSize大：说明目的数据库应用事务慢；源服务distance\_behide大：说明源服务事务分析慢）
* <Schema>.<Table> 表示当前对schema为RDISA\_T的表EMP共有多少个操作。按照表统计操作数， 目的是能够帮助用户分析出 ”热表“， 在模块划分的时候可以更加合理。但是这个统计对性能有一定的影响。

### 属性

Target Module中也有属性配置，各项配置及意义如下：

|  |  |
| --- | --- |
| Datasource\_name | Target Module中使用的数据源名，事务将通过这个数据源应用的目的数据库 |
| suspend\_on\_exception | 当Target Module执行遇到异常时是否suspend该服务 |
| auto\_recover | 当遇到Message Queue或者数据库连接中断时，是否自动恢复,当Target Module配置为suspend\_on\_exception=true,并且service因为异常而suspend时，此项无效，不自动恢复。 |
| mail\_notification | Mail通知配置 |
| Service\_log\_level | 服务日志的输出级别 |
| Associated\_mq.url | 用于连接消息中间件的URL.目前默认提供activeMQ作为消息中间件。用户可以多种方式连接到消息中间件。例如：  server.home/conf/agent.properties有如下配置：  mq.url=vm:(broker:(tcp://localhost:8080)?persistent=true)?marshal=false  表明target Server 用本地VM协议连接到activeMQ，并且他们运行在同一个进程中，同时activeMQ对其他客户端开放tcp协议访问。此时Target Server可使用tcp协议连接到activeMQ, server.home/conf/agent.properties 应作如下配置：  mq.url=tcp://localhost:8080  在linux环境网络环境中应该使用真实IP地址，不要使用127.0.0.1  activeMQ亦可运行于单独的进程中，以及支持http等多种协议。详细了解可见 http://activemq.apache.org/ |
| mq.username | 连接到消息中间件的用户名 |
| mq.password | 连接到消息中间件的密码 |
| Associated\_jms\_destination | 接收消息的消息队列名称 |
| tx\_branch\_size | 用于Target Server的并发应用事务策略。当分析得一个事务可以并发应用的情况下，Target Server将使用多线程多连接并发执行。  tx\_branch\_size=0 表示不并发执行  tx\_branch\_size>0 表示事务中的SQL操作将以此大小为单位进行分割并发执行。若系统得不到分割完所得的连接数，那么Target Server将用可得到的连接数进行重新分割，然后并发执行。 |
| tiny\_tx\_merge | 用于小事务频繁的业务。 Target Server将小事务合并在一起提交。  tiny\_tx\_merge=0 表示不进行事务合并  tiny\_tx\_merge>0 表示若收到的一组事务的平均事务大小小于此值，那么这一组事务将作为一个事务提交。 |

### 事务拦截器

RDI Transaction Interceptor 用于Target Module，在配置Target Module时可以对其进行设置，以实现同步数据之外的额外工作，类似AOP功能，Transaction Interceptor也是依配置的次序执行。到目前为止，API尚未开放。RDI内部已经实现了5个interceptor。

* TableMirroringInterceptor
* SimpleTransactionInterceptor
* SetClauseTrimmer
* WherePKTransactionInterceptor
* DigestTransactionInterceptor

Transaction Interceptor通用属性：

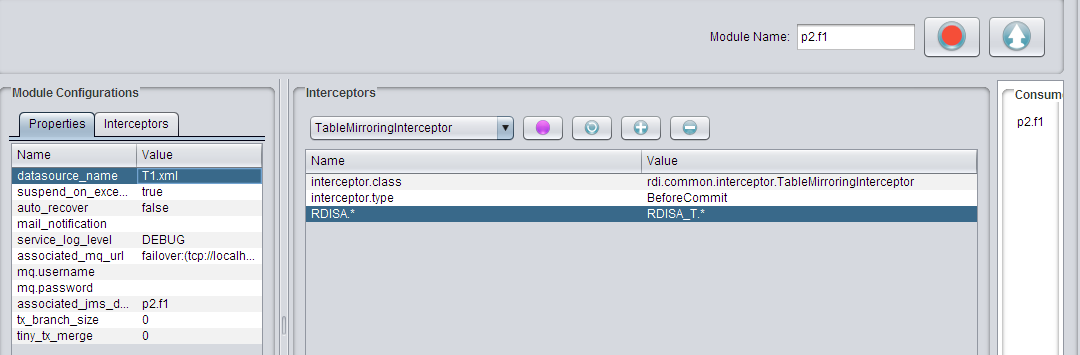
interceptor.class TransactionInterceptor的完整classname

interceptor.type 分为BeforeCommit, AfterCommit两种,分别为事务提交之前和之后被调用。

以下分别介绍TransactionInterceptor。

1. **TableMirroringInterceptor**

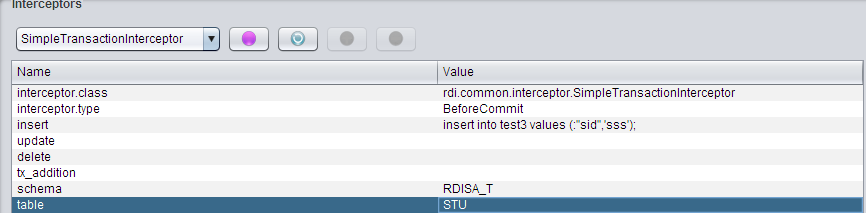
该拦截器是表镜像事务拦截器，是Target Module中基本的拦截器，而且在配置Target Module时会默认生成一个该拦截器，用于指定源表到目地表建立的镜像。其默认的镜像是”源Schema.\*→ 目的Schema.\*”,当然用户可以根据自己的实际应用具体指定该镜像。界面如下图：



点击保存当前的事务拦截器设置，点击新建一个事务拦截器。

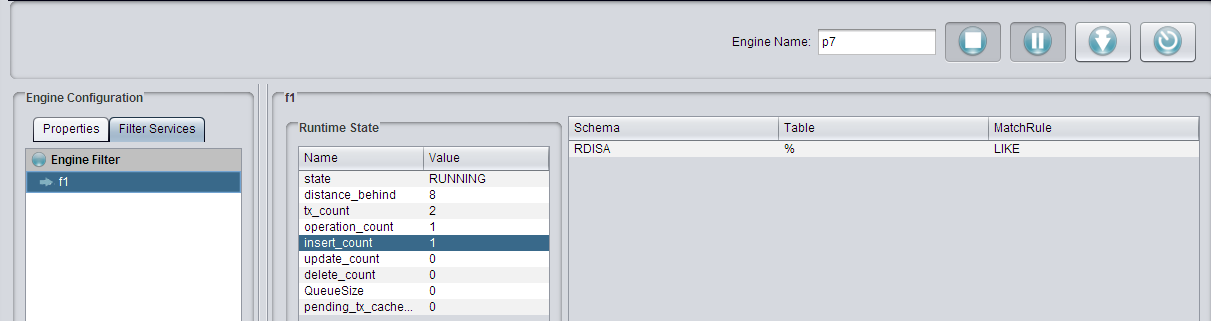
1. **SimpleTransactionInterceptor**

功能： 用于给事务增加额外操作，如图：



schema为目的端的shecma，table是指对该schema的那个表操作时，会执行附加操作（insert，update，delete，tx\_addtion）这里的schema不可以使用通配符“%”；

在设两级引擎时，第一级用通配符，第二级用具体的表结构，第二级可以用多个过滤器；目前第二级不支持通配符，如下配置。



如果这样配置，那么源端挖掘的事务传不到目的端数据库上，都被过滤掉

**# 对每个insert语句增加的操作**

# :"<column\_name>" 用户引用对此列insert的新值,可理解为此列的新值

insert=delete from table2;insert into table2 values( :"ID" ,'zzz');

**#对每个update语句增加的操作**

# ::"<column\_name>" 引用where子句中相应列名对应的值,可理解为此列的旧值

# :"<column\_name>"引用set子句中相应列名对应的值,可理解为此列的新值

update=delete from table2 where id= ::"ID";insert into table2 values( :"ID", :"NAME")

**# 对每个delete语句增加的操作**

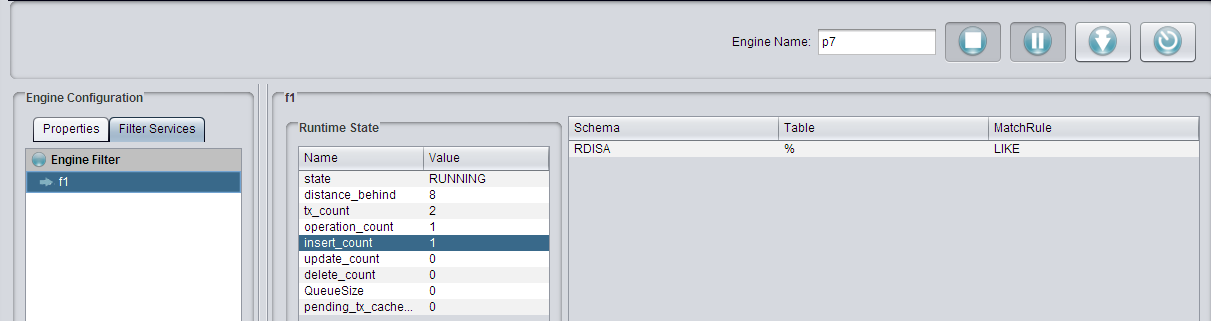
# ::"<column\_name>" 引用where子句中相应列名对应的值,可理解为此列的旧值

delete=delete from table2 where id = ::"ID"

**# 对整个事务增加的操作，这里无法使用引用的形式**

tx\_addition = insert into table2 values( 999 ,'zzz');

注意这里在配置该项时，设置源端数据源中的两级过滤器是，第一级可用通配符，第二级用具体的表结构，第二级可以用多个过滤器；但目前第二级不支持通配符（对其他配置可以，如对columnfilter），如下是错误的用法。



如果这样配置，那么源端挖掘的事务传不到目的端数据库上，都被过滤掉。

1. **SetClauseTrimmer**

功能：对于update语句，如果set子句中的新值与where子句中的列对应的旧值相同，那么将set子句中的这个列去掉，如果set子句中的新值与where子句中的列对应的旧值全部相同，那么这个sql语句会从事务中删除。

在事务日志中，提交的sql会解析为具体的带有每个属性的原子操作，这样就能够检查update中set子句的属性是否出现在where子句中。setClauseTrimmer就是把在where中出现的属性，如果在set中某个属性出现的值和where同属性的值一样，那么就删除set中的这个属性，如果set中的所有属性的值都出现在where中，那么就把这个操作删除。

Sql：update stu set sname = 'wangxiao', grade = grade+10 where sname = 'wangxiao' ；

在目的端会会去掉sname属性，然后执行。

Sql：update stu set sname = 'wangxiao', grade = 106 where sname = 'wangxiao' and grade =106;

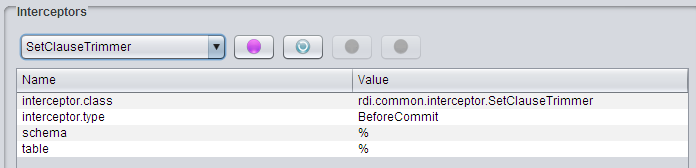
删掉整个sql，不用执行。

#支持通配，对符合此支持like语法，如果对所有的表都做此优化的话可以用%

schema=%

table=%

其操作界面如下：



1. **WherePKTransactionInterceptor**

功能：如果表有PK,那么将where子句中的其他字段条件去掉，只剩下PK字段条件。

# PK字段，双引号, 可以包含多列, 用逗号分割

pk.columns="PK\_VOUCHER"

# 对 RDISA.TABLE1使用这个interceptor,对其他表不使用

schema=RDISA，注意这里不能用通配符（%）

table=GL\_VOUCHER

对于Sql= update stu set grade = 200 where sid like '%1' and grade =106;

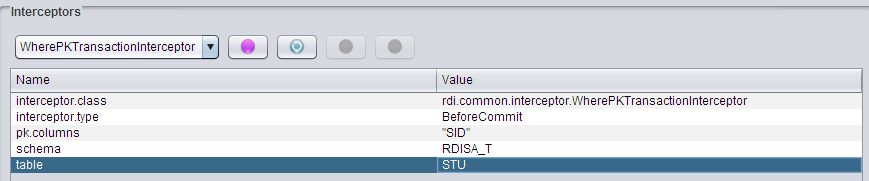
那么在数据库日志中会具体变为原子操作，可能这样一条语句变为了n条，其中一条假设是

update stu set grade = 200 where sid=11 and grade =106;

那么在目的端执行时会去掉where子句中主键以外所有属性,如grade，修改后变为：

update stu set grade = 200 where sid like '%1' and grade =106;

其操作界面如下：



注意这里的schema不能用通配符。

1. **ColumnFilter**

功能：对insert和update操作，用户可以根据需要配置只更新特定的列。

#列名用双引号, 可以包含多列, 用逗号分割

insert.filter.columns="AAA"

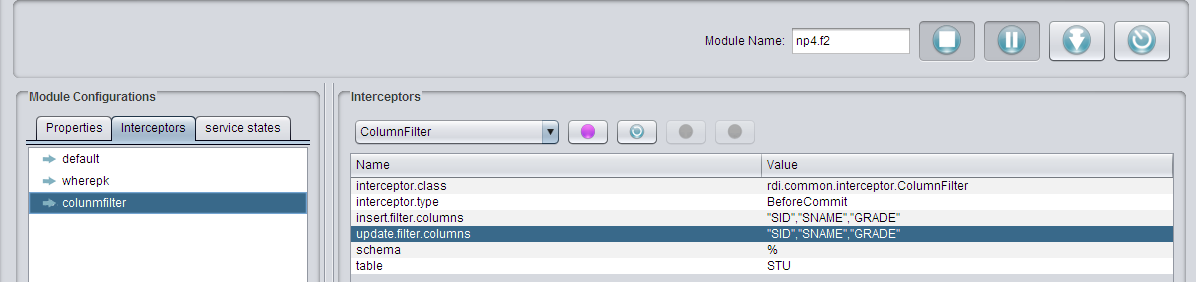
update.filter.columns="AAA"

# 必须属性

schema=RDISA

table=AAA

其操作界面如下：



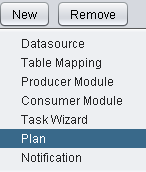
这样做是因为目的端数据库不一定需要源端数据库的所有数据，通过设置可以选定某些关心的列。（这里schema可以用通配符）。

# RDI计划脚本详解

***本章内容概要：***

* ***命令简介***
* ***上下文属性***
* ***用脚本设计器生成脚本***

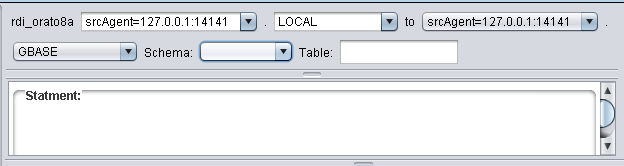
RDI计划脚本是一系列命令和上下文环境组织成的脚本，这个脚本可以按照用户的设置定期自动执行。当用户有大量原始数据需要花费很多时间导入时用户就可以将其组织成计划脚本，设置在特定时间来执行。新建计划脚本：



### 命令简介

RDI中脚本计划命令的功能介绍：

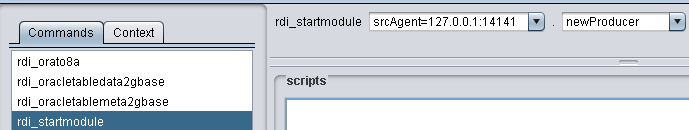
1. **rdi\_orato8a**



功能：用于从oracle 表导入特定数据（用查询语句SQL statement指定）到目的端数据仓库的特定表中。

参数：源agent，源数据源，目的agent，目的数据源，目的scheme，目的表名，查询语句SQL statement

1. **rdi\_startmodule**



功能：启动RDI module

参数：两个参数，一个是agent，一个是module名称

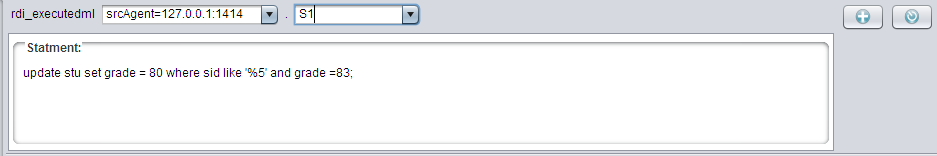
1. **rdi\_stopmodule**



功能：停止RDI module

参数： 同rdi\_startmodule

1. **rdi\_executedml**

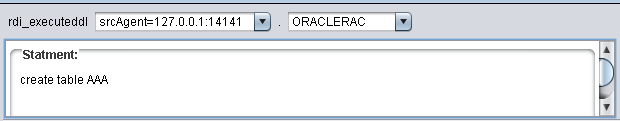


功能：在数据源执行DML操作

参数：agent， 数据源， DML语句

执行dml一般用来insert ，update，delete，不适合用来查询。

1. **rdi\_executeddl**



功能：在数据源执行DDL操作

参数：agent， 数据源， DDL语句

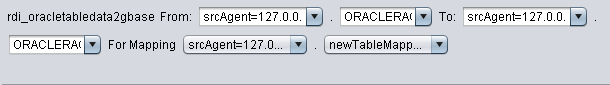
1. **rdi\_oracletablemeta2gbase**



功能：在目的端数据仓库中创建表映射配置中定义的表结构

参数：源agent，源数据源，目的agent，目的数据源，mapping所在的agent，table mapping配置。

1. **rdi\_oracletabledata2gbase**

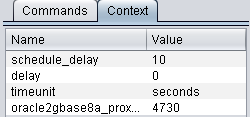


功能：在目的端数据仓库中导入表映射配置中定义的表数据

参数：源agent，源数据源，目的agent，目的数据源，mapping所在的agent，table mapping配置。

### 上下文属性

每个计划脚本执行需要有相应的上下文属性，如图：

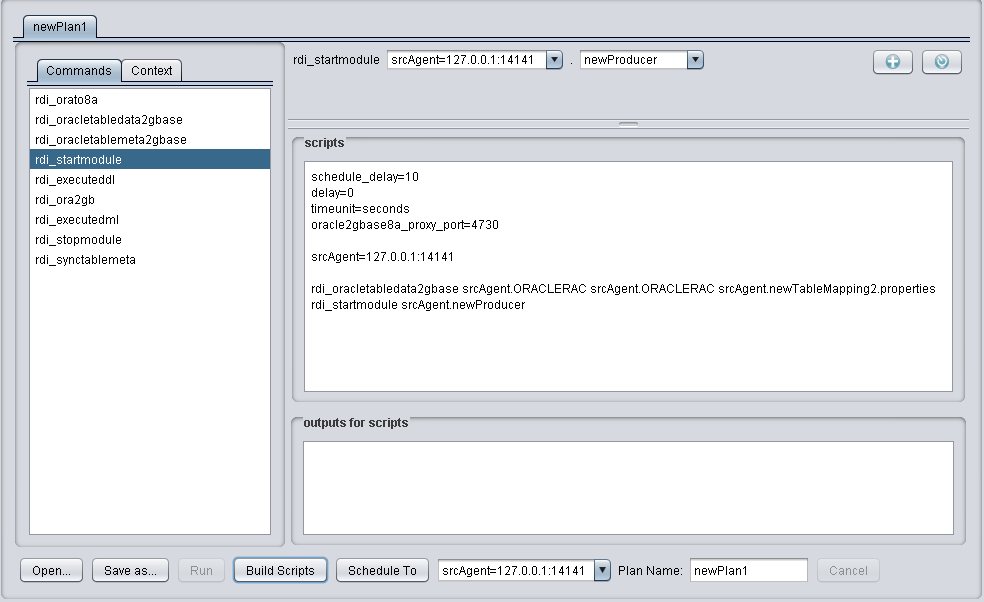


具体意义如下：

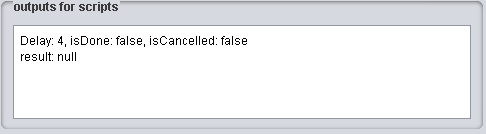
|  |  |
| --- | --- |
| Schedule\_delay | 脚本计划延后多少时间开始执行 |
| Delay | 脚本计划循环多次执行时，每次执行之间的时间间隔；如果为0，则表示只执行一次 |
| Timeunit | 脚本计划执行的时间单位 |
| Oracle2gbase8a\_proxy\_prot | 数据仓库导入代理器的端口号 |

### 用脚本设计器生成脚本

用户可以用脚本设计器方便的生成脚本，首先在Commands列表框中选择命令，然后在参数面板上选择命令参数，点击界面右上侧的  按钮，向脚本中加入该命令，然后在点击  按钮可以吧上下文属性加入脚本中，点击按钮将脚本部署到选定的RDI Server上，如图所示：

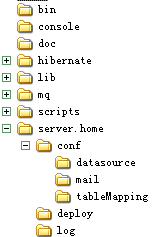


在左边的资源导航栏的plan下双击此plan，如图，可以显示该plan，在右下方的”output for scripts”框中可以看到当前计划的执行状况，如图：



# RDI目录和系统配置文件

RDI安装以后目录结构如下：



bin下是rdi启动脚本

console是管理控制台，可以拷贝到任意目录下独立运行，执行console.bat脚本即可。

doc下有RDI使用手册

server.home/conf/agent.properties是RDI的系统配置文件，包含配置项：

|  |  |
| --- | --- |
| 配置项名称 | 配置项描述 |
| agent.name | RDI Server 节点的名称 |
| com.sun.management.jmxremote.address | RDI Server的IP地址 |
| com.sun.management.jmxremote.port | RDI Server客户端端口号（JMX） |

server.home/conf/datasource 下保存了用户部署的数据源配置。

server.home/conf /mail 下保存了用户部署的邮件通知配置文件。

server.home/conf /tableMapping 下保存了用户部署的表映射配置文件。

server.home/deploy下保存了用户部署的RDI Module配置文件。

server.home/log下保存了系统日志文件和任务级别的日志文件，任务级别的日志文件会在任务名称的目录中。