IBM Security Directory Integrator
V 7.2.0.1

用户指南

S151-1410-03

IBM Security Directory Integrator
V 7.2.0.1

用户指南

S151-1410-03

注意

在使用本资料及其支持的产品之前，请阅读第 219 页的『声明』中的常规信息。

版本声明

注: 此版本适用于 *IBM Security Directory Integrator* 许可程序 **(5724-K74) V7.2.0.1** 及所有后续发行版和修订版，直至新版本中另有声明为止。

目录

关于本出版物

本出版物包含使用 IBM® Security Directory Integrator 中的组件来开发解决方案所需的信息。

IBM Security Directory Integrator 组件旨在供负责维护用户目录及其他资源的网络管理员使用。假设您具有安装和使用 IBM Security Directory Integrator 和 IBM Security Directory Server 的实际经验。

本信息也适用于负责使用 IBM Security Directory Integrator 开发、安装和管理解决方案的用户。该出版物的读者必须熟悉那些与已开发解决方案相连的系统的概念和管理。这些系统可能包括但不限于以下一个或多个产品、系统和概念，具体取决于解决方案:

- IBM Security Directory Server
- IBM Security Identity Manager
- IBM Java™ 运行时环境 (JRE) 或 Oracle Java 运行时环境
- Microsoft Active Directory
- Windows 和 UNIX 操作系统
- 安全管理
- 因特网协议，包括超文本传输协议 (HTTP)、安全超文本传输协议 (HTTPS) 和传输控制协议/因特网协议 (TCP/IP)
- 轻量级目录访问协议 (LDAP) 和目录服务
- 受支持的用户注册表
- 认证和授权概念
- SAP ABAP 应用程序服务器

访问出版物和术语

请阅读 IBM Security Directory Integrator V7.2.0.1 库及可在线访问的相关出版物的描述。

该部分提供以下内容:

- 『IBM Security Directory Integrator 库』 中的出版物列表。
- 第 viii 页的『在线出版物』 的链接。
- 第 ix 页的『IBM Terminology Web 站点』 的链接。

IBM Security Directory Integrator 库

IBM Security Directory Integrator 库中提供了以下文档:

- *IBM Security Directory Integrator V7.2.0.1 Federated Directory Server 管理指南*

  包含有关使用 Federated Directory Server 控制台来设计、执行和管理数据集成解决方案的信息。另外，还包含有关使用跨域身份管理系统 (SCIM) 协议和身份管理界面的信息。

- 《*IBM Security Directory Integrator V7.2.0.1 入门指南*》

  包含 IBM Security Directory Integrator 的简要教程和介绍。包含用于创建 IBM Security Directory Integrator 的交互与实际操作学习的示例。

- 《*IBM Security Directory Integrator V7.2.0.1 用户指南*》

  包含关于使用 IBM Security Directory Integrator 的信息。包含关于使用 Security Directory Integrator 设计器工具（配置编辑器）来设计解决方案或从命令行运行现成的解决方案的指示信息。还提供了关于界面、概念和 AssemblyLine 创建的信息。

- 《*IBM Security Directory Integrator V7.2.0.1 安装和管理员指南*》

  包含了关于安装、从前版本迁移、配置记录功能和 IBM Security Directory Integrator 远程服务器 API 底层安全模型的完整信息。包含关于如何部署和管理解决方案的信息。

- 《*IBM Security Directory Integrator V7.2.0.1 参考指南*》

  包含关于 IBM Security Directory Integrator 的独立组件（连接器、函数组件、解析器和对象等 – AssemblyLine 的构建模块）的详细信息。

- 《*IBM Security Directory Integrator V7.2.0.1 问题确定指南*》

  提供关于 IBM Security Directory Integrator 工具、资源和技术的信息，这些信息可以协助确定和解决问题。

- 《*IBM Security Directory Integrator V7.2.0.1 Message 指南*》

  提供与 IBM Security Directory Integrator 关联的所有参考、警告和错误消息的列表。

- 《*IBM Security Directory Integrator V7.2.0.1 Password Synchronization Plug-ins 指南*》

  包含安装并配置以下五个 IBM Password Synchronization Plug-ins 的完整信息: Windows Password Synchronizer、Sun Directory Server Password Synchronizer、IBM Security Directory Server Password Synchronizer、Domino® Password Synchronizer 以及 Password Synchronizer for UNIX 和Linux。 还提供了针对 LDAP 密码存储和 JMS 密码存储的配置指示信息。

- *IBM Security Directory Integrator V7.2.0.1 发行 说明*

  描述文档中未包含的关于 IBM Security Directory Integrator 的新功能和最新信息。

在线出版物

IBM 发行产品时会发布产品出版物，并会在以下位置更新出版物:

IBM Security Directory Integrator 库
产品文档站点 (http://www-01.ibm.com/support/knowledgecenter/SSCQGF/welcome) 显示该库的欢迎页面和导航。

IBM Security Systems Documentation Central
IBM Security Systems Documentation Central 提供了所有 IBM Security Systems 产品库的按字母排序的列表以及每个产品特定版本的联机文档。

IBM 出版物中心
“IBM 出版物中心”站点（ http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) 提供了可帮助您查找所有所需 IBM 出版物的定制搜索功能。

相关信息

以下位置提供了与 IBM Security Directory Integrator 相关的信息:

- IBM Security Directory Integrator 使用 Oracle 的 JNDI 客户机。关于 JNDI 客户机的信息，请参阅 *Java Naming and Directory Interface™ Specification*，地址为 http://download.oracle.com/javase/7/docs/technotes/guides/jndi/index.html 。
- 在以下地址可以找到可能帮助解答与 IBM Security Directory Integrator 相关的问题的信息: https://www-947.ibm.com/support/entry/myportal/over-accesspubsview/software/security_systems/tivoli_directory_integrator。

IBM Terminology Web 站点

IBM Terminology Web 站点将产品库的术语合并在一处。您可以通过 http://www.ibm.com/software/globalization/terminology 访问 Terminology Web 站点。

辅助功能选项

辅助功能可以帮助身体有残疾的用户（例如行动不便或有视力障碍）成功使用软件产品。借助本产品，您可以使用辅助技术来收听和导航界面。另外，您还可以使用键盘来操作图形用户界面中的所有功能，而无需使用鼠标。

有关更多信息，请参阅 *配置 Directory Integrator* 中的“辅助功能选项附录”。

技术培训

有关技术培训信息，请参阅以下 IBM Education Web 站点，地址为 http://www.ibm.com/software/tivoli/education。

支持信息

IBM 支持为与代码相关的问题和程序、短期安装或用法问题提供帮助。您可以直接访问 IBM 软件支持站点 (http://www.ibm.com/software/support/probsub.html)。

故障诊断 提供了以下详细信息:

- 联系 IBM 支持之前要收集的信息。
- 联系 IBM 支持的各种方法。
- 如何使用 IBM 支持助理。
- 自行隔离和修复问题的指示信息和问题确定资源。

良好安全实践的声明

IT 系统安全性是指通过预防、检测以及对来自企业内外的非法访问作出响应来保护系统和信息。非法访问可能会导致信息遭到更改、破坏、盗用或滥用，或系统遭受损坏或滥用（包括用于攻击他人）。不应将任何 IT 系统或产品视为完全安全，也没有任何产品、服务或安全措施可完全有效地阻止非法使用或非法访问。 IBM 系统、产品和服务旨在成为全面安全方法的组成部分，全面安全方法必须包含其他操作过程，可能需要

使用其他系统、产品或服务才能达到最佳效果。IBM 不保证任何系统、产品或服务免受任何一方恶意或非法行为的影响，或使您的企业免受任何一方恶意或非法行为的影响。

第 1 章 常规概念

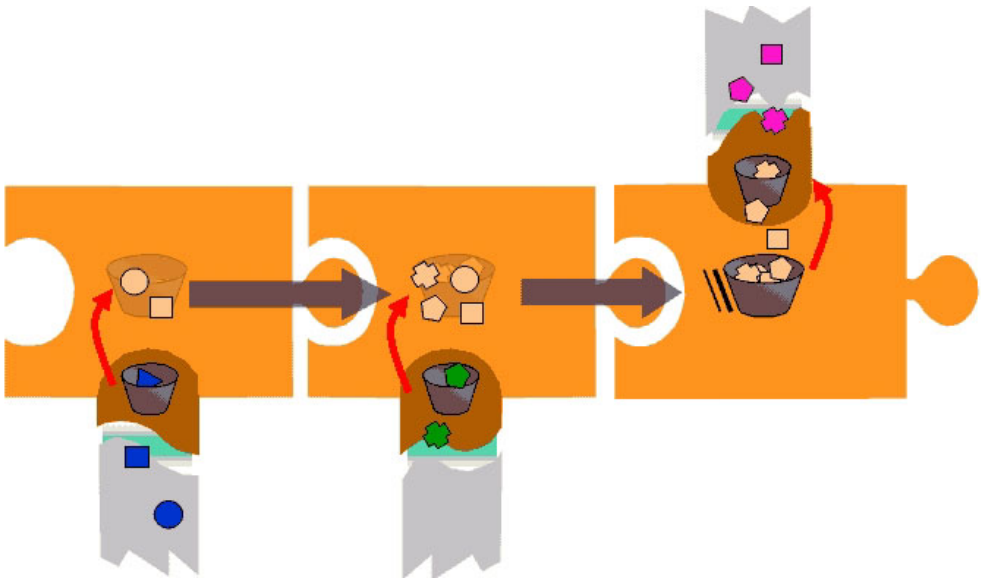
本章节介绍了 IBM Security Directory Integrator 中的部分基本概念，以及可用于构建解决方案的体系结构元素及其特征和行为。

AssemblyLine

AssemblyLine（AL）是一组串在一起来移动和变换数据的组件；它描述数据将要经过的“路径”。通过该过程处理的数据表示为*条目*对象。AssemblyLine 在它的每次循环中一次处理一个条目。它是 IBM Security Directory Integrator 中的工作单元，通常表示从一个或多个数据源到一个或多个目标的信息流。

概述

构成 AssemblyLine 的有些组件从一个或多个*已连接系统*检索数据 – 按此方式获取的数据视为 AL 的“订阅源”。要处理的数据会每次一个*条目*传入到 AL 中，其中这些条目会携带具有来自目录条目、数据库行、电子邮件、*Lotus® Notes®* 文档、记录或类似数据对象的*值*的属性。每个条目都包含了*属性*，这些属性用于保存从源系统中的字段或列读取的数据值。对这些属性进行重命名，重新格式化或将其作为 AL 中从一个组件到下一个组件的处理流计算。可以从其他源“连接”新信息，并且可将全部或部分已变换数据按要求写入目标存储或发送到目标系统。因此可使用下图对此进行解释:



在此图中，绘制大量拼图块集合作为 AssemblyLine，从下面进入的灰色流中最左边的蓝色点和正方形作为输入流的原始数据，右上方的紫色点作为输出流的数据输出。与拼图和其中的存储区交叉的暗橙色元素表示*解析器*，它将原始数据变成结构化数据，然后可以让该数据开始沿着 AssemblyLine（存储区中的浅色元素）向下流。中间的拼图块描绘了正在读取已结构化数据（例如，从数据库）的连接器。

数据以某种输入方式，使用第 3 页的『连接器』从已连接系统进入 AssemblyLine，然后以某种输出方式，使用连接器输出到一个或多个已连接系统。

还可以从面向记录的系统（例如数据库或消息队列）读取数据：在这种情况下，可以很容易将输入中的各列映射到得到的工作条目中的属性，该工作条目绘制为左边拼图中的“存储区”。或者，可以从数据流（如文件系统中的文本文件、网络连接等）读取数据。在这种情况下，解析器可以作为连接器的前缀，以便理解输入流，并将输入流分解，然后可以分配给工作条目中的属性。

一旦第一个连接器完成它的工作，信息存储区（“工作条目”，恰当的说，称为*工作*）将沿着 AssemblyLine 传给下一个组件 – 在本图中，为另一个连接器。由于第一个连接器中的数据可用，因此它现在可用作在第二个已连接系统中检索或查询数据的关键信息。一旦找到相关数据，就可以将其合并到*工作*中，从而补充仍然来自第一个连接器的数据。

最后，合并的数据此时将以某种输出方式沿着 AssemblyLine 传给第三个拼图块或连接器，该过程将数据输出到已连接系统。如果已连接系统是面向记录的，那么*工作*中的各种属性仅映射到记录中的列；如果已连接系统是面向流的，那么解析器可以执行必要的格式化。

可以在 AssemblyLine 中随意插入其他组件（如第 19 页的『脚本组件』和第 18 页的『函数』），从而对*工作*中的数据执行操作。

请记住，AssemblyLine 设计和优化为一次处理一个项，这一点很重要。但是，如果您想要执行多个更新或多个删除（例如，一次处理多个项），则必须编写 AssemblyLine 脚本以执行该操作。如果需要，此类处理可以使用 JavaScript、Java 库和标准 IBM Security Directory Integrator 功能实现，如将数据合并到已排序的数据存储中（如使用 JDBC 连接器），然后重新读取它，并使用第二个 AssemblyLine 对它进行处理。

可使用 IBM Security Directory Integrator 配置编辑器（CE）构建、配置和测试 AssemblyLine，请参阅第 67 页的第 3 章，『配置编辑器』了解更多信息。AssemblyLine 在配置编辑器中具有“数据流”选项卡。该选项卡是保存组成该 AL 的组件列表的位置。

AL 中的所有组件都自动注册为脚本变量。因此，如果您有一个名为 ReadHRdump 的连接器，那么您可以使用 *ReadHRdump* 变量直接从脚本中访问它及其方法。因此，就像对脚本变量一样，您将希望对 AL 组件命名。命名时请按以下规则：仅使用字母数字字符；名称不要以数字开头；不要使用特殊的本国字符（例如，å、ä）、分隔符（除下划线“_”以外）、空格等。

总是会有备用的方法来访问 AL 组件（例如，`task.getConnector()` 函数），但是有意识地遵守命名约定始终是明智的。

在 IBM Security Directory Integrator 中启动 AssemblyLine 是代价相当大的操作，因为该操作涉及创建新的 Java 线程，并且通常会与一个或多个数据源建立连接。请仔细考虑是否能让您的解决方案设计处理较少（而不是较多）的完全不同的 AssemblyLine，其中每条 AssemblyLine 将承担更多的工作；例如，通过使用分支或 Switch 来定义由单个 AL 处理的多个操作。请注意：每个操作仍然可以作为单条 AssemblyLine 实施，但是它们可被“现成地”嵌入通过使用 AL 连接器或 AL 函数向它们分派工作的单个 AL 中。该方案还可以允许您利用诸如全局连接器池之类的功能部件来管理资源的使用并提升性能和可伸缩性。

组件

AssemblyLine 可以包含以下组件:

- 『连接器』
- 第 18 页的『函数』
- 第 19 页的『脚本组件』
- 第 20 页的『属性映射』
- 第 22 页的『分支组件』

此外，连接器可以配置第 25 页的『解析器』；同时，在“系统”、“配置”、“AssemblyLine”、“属性映射”和“属性”级别，存在用于配置第 21 页的『Null 行为』的选项。

访问 AssemblyLine 内部的 AL 组件

每个 AL 组件都以具有为该组件选择的名称的预注册脚本变量提供。

请注意，您可以通过对诸如 `system.getConnector()` 之类的函数进行脚本调用来动态装入组件，但这不适用于缺乏经验的用户。[1]

AssemblyLine 参数传递

有三种方式将数据放入 AssemblyLine:

- 在 AssemblyLine 中（例如，在 Prolog 脚本中）生成您自己的初始条目。
- 从一个或多个迭代器供给 [2]。
- 从使用 AL 连接器或 AL 函数组件，或使用 API 调用的另一个 AssemblyLine 带参数启动 AssemblyLine。

如果您想要用来自其他 AssemblyLine 的参数启动 AssemblyLine，则您有两个选择:

- 使用*任务调用块*（TCB），这是首选的方法。关于更多信息，请参阅第 58 页的『任务调用块（TCB）』。此部分还讨论了动态禁用或启用 AssemblyLine 组件的方法。
- 直接提供“初始工作条目”；请参阅第 6 页的『提供初始工作条目（IWE）』了解详细信息。

注: 提供这些选项是为了保持与先前版本的兼容性。

连接器

连接器用于访问并更新信息源。连接器的工作就是对正在运行的字段分级，从而使您不必处理使用各种数据存储、系统、服务或传输的技术细节。同样地，每个类型的连接器都设计为使用特定的协议或 API，处理数据源访问的细节以便您可以专注于数据操作和关系，以及定制诸如过滤和一致性控制的处理。

---

1. 从该调用中获取的连接器对象是*连接器接口*对象，并且是 AssemblyLine 连接器中特定于数据源的部分。当更改任何连接器的类型时，实际上是在换出其数据源智能（连接器接口），该数据源智能提供在特定系统、服务或数据存储器上访问数据的功能。AssemblyLine 连接器的大多数功能（包括“属性映射”、“链接条件”和挂钩）由内核提供，并在转换连接器类型时保持不变。

2. *迭代器*是连接器在 Iterator 方式的简写

概述

连接器用于抽取某个系统或存储器的详细信息，从而为您提供相同的访问功能部件集合。这使您可以通过一致并可预见的方式使用多种不同技术和格式。典型的 AssemblyLine（AL）具有一个提供输入的连接器和至少一个用于写出数据的连接器。

有两种类别的连接器:

- 第一种类别是数据内容的传输和结构对连接器都是已知的；也就是，可以使用知名的 API（如 JDBC 或 LDAP）来查询或检测数据源的模式。
- 第二种类别是传输机制已知，但内容结构未知 – 通常看起来像数据流。该类别需要*解析器*（请参阅第 25 页的『解析器』和 *参考* 中的“解析器”部分）来解析或生成内容结构，以供 AssemblyLine 正常运行。

富连接器库是 IBM Security Directory Integrator 的一个优势。您可以在 *参考* 中找到 IBM Security Directory Integrator 包含的所有连接器的列表。但是您也可以使用 JavaScript 或甚至 Java 编写您自己的连接器；请参阅 *参考* 中的“Implementing your own Components”。

每个连接器都为特定的协议、API 或传输而设计，并处理所连接系统的本机类型和 Java 对象之间的已编组数据。与其他组件不同，连接器具有“方式”设置，用于确定该连接器如何访问其连接的系统；请参阅第 5 页的『连接器方式』了解更多信息。每个连接器仅支持适合于其所连接系统的方式子集。例如，“文件系统连接器”只支持单独一种输出方式（AddOnly），而不支持 Update、Delete 或 CallReply。使用连接器时，必须首先查阅该组件的文档以获取受支持方式的列表。

注: AssemblyLine 可以根据需要，由很多或很少的连接器（和其他组件）构成以实现您的特定数据流。系统中没有限制。然而，最好的做法是使 AssemblyLine 尽量简单以便可维护性达到最大程度。

当您为 AssemblyLine 选择连接器时，将显示一个对话框，它使您能够选择您想要从中*继承*的连接器类型。使用 IBM Security Directory Integrator 时，继承是一个重要的概念，因为解决方案中包含的所有组件都继承另一个组件的部分或全部特性，其中另一个组件是一种基本类型的组件，或者是来自预配置的组件（您工作空间的“资源”部分的“连接器”、“解析器”、“函数”等）库中的组件。

当在 AL 中使用时，连接器提供“初始化”选项来控制对组件进行设置的时间；例如，建立连接、资源绑定等。在缺省情况下，所有连接器在 AssemblyLine 启动时进行初始化: *AL 启动阶段*。

Server 或 Iterator 方式的连接器“供给”AssemblyLine，并负责在 AL 进行的每次循环中为 AL 提供新的工作条目。工作条目在“流程”部分（按照您实现的任何分支逻辑）从一个组件传递到另一个组件，直到“流程”结束为止。此时，循环结束行为出现，例如，迭代器从其源获取下一个条目，然后将其传给“流程”部分开始新的循环。

当“供给”部分的迭代器实际地驱动“流程”时，“流程”部分的迭代器将只获取下一个条目并为到工作条目的输入映射提供其数据属性。

注: 可以将 Iterator 方式的连接器放置到“流程”部分。同样，迭代器可以按其在“供给”中相同的方式工作: AL 启动期间将初始化迭代器（包括通过 `selectEntries` 调用构建

其结果集），迭代器将在每个 AL 循环中检索一个条目（getNextEntry）。但是，“流程”部分的迭代器不会驱动 AL 自身，因为它将在“供给”部分执行。

“供给”部分的行为在 Server 方式和 Iterator 方式下是不同的: 迭代器期望成为在 AL 中运行的第一个组件，并且如果工作条目已不存在，则它将只读取它的下一个条目。如果 AL 传递了“初始工作条目”，则迭代器在该首个周期中不读取任何数据。这还意味着迭代器将按顺序运行，此时一旦首个迭代器到达了数据的结尾，且未返回任何值（null），则从第二个迭代器开始返回条目。

另一方面，Server 方式使得连接器启动服务器 *listener* 线程（例如，IP 端口或事件通知回调），然后将控制传递到下一个“供给”连接器。

当使用 AssemblyLine 函数组件（AL FC）调用 AL 时，如果使用手动循环方式，那么每当 FC 执行调用时仅使用“流程”部分的组件。

IBM Security Directory Integrator 导航器的工作空间中有一个**连接器**文件夹，您可以在其中维护已配置连接器的库。这也是定义连接器池的位置。

连接器方式

AssemblyLine 连接器的方式定义了该连接器在数据流中所扮演的角色，并控制了 AssemblyLine 的自动行为如何驱动组件。它确定了是否从输入源中读取和/或写入该输入源。

可以将连接器设置为以下这些标准方式中的一种:

- Iterator
- Lookup
- AddOnly
- Update
- Delete
- CallReply
- Server
- Delta

这些方式会在下面几节中讨论。对于连接器方式行为的详细描述，以及对 AssemblyLine 的一般描述，请参阅 *参考* 中的“AssemblyLine and Connector mode flowcharts”。

Iterator 方式:

Iterator 方式的连接器用于扫描数据源和抽取其数据。Iterator 方式的连接器实际上通过数据源条目进行迭代、读取其属性值，并且将每个条目传送到 AssemblyLine“流程”部分的组件中（每次传送一个）。处于 Iterator 方式的连接器一般被称为“Iterator 连接器”或只是 Iterator。

AssemblyLine（除了那些通过 IWE 调用的以外; 请参阅第 6 页的『提供初始工作条目（IWE）』）通常包含至少一个 Iterator 方式的连接器。迭代器通过构建工作条目并将这些工作条目传递到 “AL 流”部分来为 AssemblyLine 提供数据。

流程部分的组件按顺序启动，从“流程”列表的顶部开始。“流程”处理完成时，控制将传回到迭代器以检索下个条目。

AssemblyLine 中的多个迭代器: 如果您具有多个 Iterator 方式的连接器，那么这些连接器将以其出现在配置（以及配置编辑器中连接器列表的供给部分）中的顺序堆栈，并且每次处理一个。因此，如果您使用两个迭代器，则第一个迭代器将从其数据源中读数据，并将得到的工作条目传递到第一个非迭代器中，直到它到达其数据集的结尾为止。当第一个迭代器处理完其输入源时，第二个迭代器开始读入数据。

初始的工作条目被视为来自隐藏迭代器（其在任何其他迭代器之前被处理）。这意味着 IWE 跳过第一个循环过程中的所有迭代器，传递至 AssemblyLine 中的第一个非 Iterator 连接器。该行为可在 *参考* 中“AssemblyLine and Connector mode flowcharts”的“AssemblyLine Flow”页面找到。

假设您有一个具有两个迭代器（**a** 在 **b** 之前）的 AssemblyLine。则第一个迭代器 **a** 将一直用到 **a** 不再返回条目为止。然后，AssemblyLine 将切换到 **b**，并忽略 **a**。如果初始工作条目（IWE）传递到该组装流水线，那么在首个循环中两个迭代器都被忽略，之后组装流水线开始调用 **a**。

有时 IWE 用于将配置参数传递给 AssemblyLine，而不是数据。然而，IWE 的出现会导致在第一个循环过程中跳过 AssemblyLine 中的迭代器。如果不希望发生这种情况，必须通过在 prolog 脚本中调用 `task.setWork(null)` 函数来清空工作条目对象。这会使第一个迭代器正常运行。

使用 Iterator 方式:

通常使用迭代器驱动 AssemblyLine。

使用处于 Iterator 方式的连接器的最常见模式是:

1. 使用配置编辑器，将处于 Iterator 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式（Iterator）和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 100 页的『输入属性映射』。

这些已映射的属性是从数据源检索得到的，它们被放置在*工作*条目中，并传给 AssemblyLine 的“流程”部分的连接器。

如果您未在 AssemblyLine 内直接创建连接器，那么为了在 AssemblyLine 中使用此连接器，请将其从 `<workspace>/Resources/Connectors` 中它所在的位置拖动到 AssemblyLine 的**供给**部分。

提供初始工作条目（IWE）: 这是使用 TCB 传递参数的另一种方法，且由于与较早版本兼容而受支持。

通过脚本中的 `system.startAL()` 调用启动 AssemblyLine 时，该 AssemblyLine 仍然可以通过在初始工作条目（通过 *work* 变量可访问它）中设置属性或属性值来传递参数。然后由您来应用这些值以设置连接器参数（例如，在 AssemblyLine **Prolog – Init** 挂钩中使用 *connectorName*.`setParam()` 函数）。

注: 必须使用 `task.setWork(null)` 调用来清除工作条目，否则 AssemblyLine 中的迭代器会通过第一个循环。

可以使用 getResult() 函数来检查 AssemblyLine 的结果，这是 AssemblyLine 停止时的**工作条目**。另请参阅 *参考* 中的“Runtime provided Connector”。

下面是在连接器参数值中用 IWE 传递的一个示例:

```
var entry = system.newEntry();
entry.setAttribute ("userNameForLookup", "John Doe");

// Here we start the AssemblyLine
var al = main.startAL ( "EmailLookupAL", entry );

// wait for al to finish
al.join();
var result = al.getResult();

// assume al sets the mail attribute in its working entry
task.logmsg ("Returned email = " + result.getString("mail"));
```

Lookup 方式:

Lookup 方式使您能够使用这些系统中属性之间的关系，连接来自不同数据源的数据。处于 Lookup 方式的连接器通常称为“查找连接器”。

要在配置编辑器中设置“Lookup 连接器”，必须告诉连接器您如何定义 AssemblyLine 中已有的数据和已连接系统中找到的数据之间的匹配。这被称为连接器的*链接条件*，每个 Lookup 连接器具有相关联的**链接条件**选项卡，在其中您可以定义规则来查找匹配的条目。关于更多信息，请参阅第 17 页的『链接条件』。

使用 Lookup 方式: 使用处于 Lookup 方式的连接器的最常见模式是:

1. 使用配置编辑器，将处于 Lookup 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式 (Lookup) 和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 100 页的『输入属性映射』。
4. 打开“连接器”配置窗口中的**链接条件**选项卡，并设置属性匹配的规则。此过程的结果将确定要从已连接系统检索的条目，在这里您有以下几种选项:
   a. 单击**添加**添加新的链接条件并从已连接的系统中选择属性、匹配运算符（例如，Equals、Begins With 等）以及将要进行匹配的工作条目属性。当连接器执行查找时，它将基于您指定的“链接条件”创建底层 API 或协议语法，使得您的解决方案独立于所用系统的类型。可以添加多个“链接条件”，它们用布尔运算符 AND 连接起来构建搜索调用。
   b. 还可以选择**通过定制脚本构建条件**，这样会打开脚本编辑器窗口，其中您可以创建自己的搜索字符串，并使用 ret.filter 对象将字符串传回到连接器。例如:

      ```
      ret.filter = "uid=" + work.getString("uid");
      ```

      请注意，表达式还可以用于动态指定任何“链接条件”使用的属性或值。请参阅第 31 页的『表达式』以获取相关信息。另见第 17 页的『链接条件』，以获取关于“链接条件”的更多详细信息。

您在“输入映射”中读取（并计算）的属性对使用 Work 条目对象的其他下游连接器和脚本逻辑是可用的。

如果您未在 AssemblyLine 内直接创建连接器，却要在 AssemblyLine 中使用连接器，请将其从 <workspace>/Resources/Connectors 中其所在位置拖动到 AssemblyLine 的**流程**部分。

AddOnly 方式:

处于 AddOnly 方式的连接器（通常称为 AddOnly 连接器）用于将新数据条目添加至数据源。

这种连接器方式几乎不需要配置。设置连接参数，然后选择（映射）要从工作条目中写入的属性。

使用 AddOnly 方式: 使用处于 AddOnly 方式的连接器的最常见和简单的模式是:

1. 使用配置编辑器，将处于 AddOnly 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式（AddOnly）和其他连接参数；必需的参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 101 页的『输出属性映射』。

如果您未在 AssemblyLine 内直接创建连接器，却要在 AssemblyLine 中使用连接器，请将其从 <workspace>/Resources/Connectors 中其所在位置拖动到 AssemblyLine 的**流程**部分。

AssemblyLine 调用连接器时，会将已映射的工作条目属性输出到已连接系统中。

更新方式:

处于 Update 方式的连接器（通常称为 Update 连接器）用于添加和修改数据源中的数据。对于从组装流水线传递的每个条目，更新连接器试图从数据源查找匹配的条目以使用接收到的条目属性值进行修改。如果找不到匹配，那么 Update 方式连接器将添加新的条目。

使用 Lookup 连接器时，必须告诉连接器连接器如何定义 AssemblyLine 中已有的数据和已连接的系统中找到的数据之间的匹配。这被称为“连接器的链接条件”，每个 Update 连接器都具有关联的链接条件（请参阅第 17 页的『链接条件』）选项卡，您可以在其中定义查找匹配条目的规则。如果未找到这样的条目，那么将新条目添加到数据源中。然而，如果找到一个匹配的条目，那么将修改该条目。如果多个条目与“链接条件”匹配，那么将调用 Multiple Entries Found 挂钩。而且，可以配置“输出映射”以指定在“添加”或“修改”操作的过程中使用哪些属性。

执行“修改”操作时，只更改数据源中那些“输出映射”中标记为“修改（Mod）”的属性。如果从 AssemblyLine 传递的条目的某个属性不具有值，那么该属性的“Null 行为”变得很重要。如果它设置为“删除”，而在修改的条目中不存在该属性，那么在数据源中将无法更改该属性。如果它设置为 *NULL*，则在修改的条目中存在该属性，但是具有空值，这意味着在数据源中已删除该属性。

Update 连接器提供的一个重要功能是“计算更改”选项。它打开时，连接器首先对照旧值检查新值，并且仅在需要的地方更新。因此您可以跳过不必要的更新，如果正在更新的特定数据源的更新操作很繁重，这样做可能很有价值。

有些更新连接器提供在进行更新时跳过不必要查询的选项。如果连接器支持该选项，那么您将在**计算更改**复选框旁边看到**跳过查询**复选框。选中该复选框后，它会更改连接器的行为，从而不执行查询就可找到与链接条件对应的条目。出于此原因，不会调用 Before/After Lookup 挂钩。也无法调用 On Multiple Entries 挂钩。选中此选项时，仅会构建搜索条件，并直接调用修改。这不同于更新方式中的缺省连接器行为：如果使用链接条件找不到条目，不会将其作为新的条目添加。

使用 Update 方式： 使用处于 Update 方式的连接器的最常见和简单的模式是：

1. 使用配置编辑器，将处于 Update 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式（Update）和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 101 页的『输出属性映射』。
4. 打开“连接器”配置窗口中的**链接条件**选项卡，并设置属性匹配的规则。这里您有两个选择：
   a. 单击**添加**添加新的链接条件并从已连接的系统中选择属性、匹配运算符（例如，Equals、Begins With 等）以及将要进行匹配的工作条目属性。当连接器执行查找时，它将基于您指定的“链接条件”创建底层 API 或协议语法，使得您的解决方案独立于所用系统的类型。可以添加多个“链接条件”，它们用布尔运算符 AND 连接起来构建搜索调用。
   b. 还可以选择**通过定制脚本构建条件**，这样会打开脚本编辑器窗口，其中您可以创建自己的搜索字符串，并使用 `ret.filter` 对象将字符串传回到连接器。例如：

      ```
      ret.filter = "uid=" + work.getString("uid");
      ```

   请注意，表达式还可以用于动态指定任何“链接条件”使用的属性或值。请参阅第 31 页的『表达式』以获取相关信息。另见第 17 页的『链接条件』，以获取关于“链接条件”的更多详细信息。

在 AssemblyLine 执行期间，具有您已选择要在输入中映射的属性的条件将添加到数据源中。

要在 AssemblyLine 中使用连接器，请将其从 `<workspace>/Resources/Connectors` 中它所在的位置拖动到 AssemblyLine 的**流程**部分。您现在可以通过将以前已映射的工作条目属性拖动到组装流水线的**属性映射**窗口中的 Update Connector 上，从而将这些属性映射到输出中。

您还可以通过右键单击此窗口中的连接器，并选择**添加属性映射项**，创建全新的属性。

注：在 Update 方式中，可以更新多个条目。请参阅 *参考* 中的“AssemblyLine and Connector mode flowcharts”。

Delete 方式：

Delete 方式连接器（通常称为删除连接器）用于从数据源中删除数据。

对于传递至删除连接器的每个条目，它都尝试在已连接系统中找到匹配的数据。如果找到单个匹配条目，请将其删除，否则，如果未找到匹配条目，将调用 On No Match 挂

钩，或在找到多个匹配时调用 On Multiple Entries 挂钩。与使用 Lookup 和 Update 方式一样，Delete 方式需要您定义规则，用以查找匹配的条目以删除。这将在连接器的“链接条件”选项卡中配置。

某些删除连接器提供在执行删除时跳过不必要查找的选项。如果连接器支持该选项，将看到“计算更改”复选框旁边的“跳过查找”复选框。选中该复选框后，它会更改连接器的行为，从而不执行查询就可找到与链接条件对应的条目。出于此原因，不会调用 Before/After Lookup 挂钩。也无法调用 On Multiple Entries 挂钩。打开该选项后，将仅构建搜索条件，并直接调用删除。

使用 Delete 方式: 使用处于 Delete 方式的连接器的最常见和简单的模式是:

1. 使用配置编辑器，将处于 Delete 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式（Delete）和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 100 页的『输入属性映射』。
4. 在“输入属性映射”选项卡中，可以从“连接器属性”列表中选择属性，然后将它们拖动到“输入映射”中。您还可以手动添加和除去属性。

   注: “输入映射”在 Delete 方式中可用以将数据源中找到的匹配条目读入 *conn* 条目对象，然后该对象就可在您的脚本中使用（例如，确定该条目是否确实需要被删除）。
5. 打开“连接器”配置窗口中的**链接条件**选项卡，并设置属性匹配的规则。此过程的结果将确定要从已连接系统检索的条目，在这里您有以下几种选项:
   a. 单击**添加**添加新的链接条件并从已连接的系统中选择属性、匹配运算符（例如，Equals、Begins With 等）以及将要进行匹配的工作条目属性。当连接器执行查找时，它将基于您指定的“链接条件”创建底层 API 或协议语法，使得您的解决方案独立于所用系统的类型。可以添加多个“链接条件”，它们用布尔运算符 AND 连接起来构建搜索调用。
   b. 还可以选择**通过定制脚本构建条件**，这样会打开脚本编辑器窗口，其中您可以创建自己的搜索字符串，并使用 `ret.filter` 对象将字符串传回到连接器。例如:

      ```
      ret.filter = "uid=" + work.getString("uid");
      ```

   请参阅第 17 页的『链接条件』以获取更多关于“链接条件”的信息。

如果您未在 AssemblyLine 内直接创建连接器，却要在 AssemblyLine 中使用连接器，请将其从 `<workspace>/Resources/Connectors` 中其所在位置拖动到 AssemblyLine 的**流程**部分。

CallReply 方式:

CallReply 方式用于请求数据源服务（例如 Web 服务），这些服务需要您发送输入参数并接收带有返回值的应答。

与其他方式不同的是，CallReply 方式可以访问输入和输出属性映射。

该连接器首先执行输出映射操作，并且在执行该操作的同时使用由输出映射操作提供的参数调用外部系统。该操作之后紧接一个输入映射操作，进而从外部系统中提取应答。

使用 CallReply 方式: 使用处于 CallReply 方式的连接器的最常见和简单的模式是:

1. 使用配置编辑器，将处于 CallReply 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。极少数连接器支持此方式。
2. 在**连接**选项卡中为此连接器设置方式（CallReply）和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置输出属性映射；请参阅第 101 页的『输出属性映射』。
4. 设置输入属性映射；请参阅第 100 页的『输入属性映射』。

请记住，输出映射中的属性是作为输入参数提供给已连接系统的；通过输入属性映射进行映射的属性包含来自已连接系统的回复。

Server 方式:

Server 方式存在于很多特选的连接器中，它设计用来提供等待入局事件的功能、分派处理事件的线程，以及将应答发回给发起方。

目前，所有 Server 方式的连接器都是基于连接的。因此，任何在其订阅源部分使用 Server 方式连接器的 AssemblyLine（AL）将初始化，然后等待传入连接（通过 TCP、HTTP 、LDAP、Web Services、SNMP 连接方式）；当启动连接之后，Server 方式连接器克隆它所属的 AL，并继续等待下一个事件（即新连接的启动）。同时，在已克隆的工作程序 AL 中，Server 方式连接器将其自身置于 Iterator 方式，并开始从连接中读取数据。然后，从连接中获取的数据将以常规的 Iterator 方式送入 AL 的其余部分（包含以下标准迭代器挂钩流），一次读取一个事件条目并将这些条目传递到其他“流程”组件进行处理，直到再无数据可读为止。在每个循环（通常只有一个循环）结束时，以 Server 方式连接器打头的 AL 将应答发回给客户机 – 除非您决定通过如 `system.skipEntry();` 之类的调用跳过应答阶段。

一旦 AL 供给完成（即，数据源已耗尽），则线程终止；此时工作程序 AL 已清除，并且如果需要的话，将通知池管理器该 AL 实例可再次使用。

原始 Server 方式连接器于此同时仍将侦听更多的连接启动。

注: 在某些极少出现的情况下，例如当您并行地向服务器发送超过 5 个客户机请求时（主要在 zOS 操作系统上），SNMP 客户机将退出，并抛出“协议数据单元（PDU）错误”的异常。但是，在更现实的实际情况中，极少会通过多个管理器（例如 SNMP 连接器）查询代理程序（例如 SNMP 服务器连接器）。

SNMP 连接器具有名为 *snmpWalkTimeout* 的尚未记录的配置参数。可以覆盖该参数的缺省值 5000 毫秒。该参数不能使用配置编辑器访问。可以使用 JavaScript 设置该参数的覆盖值。按以下格式为 snmpWalkTimeout 参数设置您需要的值:

```
thisConnector.connector.setParam("snmpWalkTimeout", "100000");
```

Server 方式和 ALPool: 可以使用 AssemblyLine 池对创建克隆 AL 的过程进行优化。检测到事件时，Server 方式连接器处理该 AL 的“流程”部分；或如果为该 AL 配置了 ALPool，则 Server 方式连接器将联系“池管理器”进程，以请求可用 AL 实例来处理该事件。

当带有 Server 方式连接器的 AssemblyLine 使用 ALPool 时，ALPool 将自始至终执行 AL 实例。在 ALPool 中的 AL 实例关闭“流程”连接器之前，ALPool 取回那些连接器，

并放入汇聚成池的连接器集合中，该连接器集合将在 ALPool 创建的下一个 AL 实例中使用。实际上，ALPool 使用 `tcb.setRuntimeConnector()` 方法。

有两个系统属性管理着连接器池的行为:

com.ibm.di.server.connectorpooltimeout

这个属性定义了在释放池化的连接器集合之前的超时时间，单位秒。

表 1. 连接器池超时属性值表

值	重要性
<0	禁用连接器合用
0	禁用超时；池连接器永不超时
>0	在池化的连接器超时之前的秒数

com.ibm.di.server.connectorpoolexclude

该属性定义不再合用的连接器类型。如果连接器的类名称出现在这个逗号分隔列表中，那么该连接器将不会包含在连接器池集合中。

当 ALPool 创建新的 AssemblyLine (AL) 实例时，它将查找可用的合用连接器集合，如果该集合存在，它将作为运行时提供的连接器提供给新的 AL 实例。这通常根据属性映射和挂钩执行等方面确保了 AL 的正确流程。

从不共享连接器。在使用时，它们只分配给一个 AL 实例。

使用 Server 方式: 使用处于 Server 方式的连接器的最常见模式是:

1. 使用配置编辑器，将处于 Server 方式的连接器添加到您的工作空间中。请参阅第 102 页的『创建连接器』。
2. 在**连接**选项卡中为此连接器设置方式（Server）和其他连接参数；必需参数标有星号（*）。有些连接器还需要在**解析器**选项卡中配置解析器。
3. 设置属性映射；请参阅第 100 页的『输入属性映射』。

这些已映射的属性是从数据源检索得到的，它们被放置在*工作*条目中，并传给 AssemblyLine 的“流程”部分的连接器。

注:

1. Server 方式连接器比较特殊，因为它们通常需要将某些信息返回到与之相连接的客户机上。由于存在此特性，因此在很多情况下通过单击**发现属性**不会得到任何有意义的模式；因此，在大多数情况下，您将需要通过选择**添加新属性**完全以手动方式设置属性映射；或者通过选择某个映射然后删除该映射，从而删除不必要的映射。对于 AL 的每个循环，按此方式映射的数据将发送回客户机；虽然前面提到过，操作的典型请求或响应方式中，通常只有一个循环。
2. 基于 TCP 协议的 Server 方式连接器具有一个称为“连接储备”的参数，它控制传入连接的队列长度。
3. 在 AssemblyLine 组件列表中，您可以看到“供给”和“流程”部分。Server 方式连接器观察到 AssemblyLine 处理中的第三个阶段（*响应*阶段）。屏幕上输出映射挂钩和响应挂钩是 Server 方式连接器自身的一部分，但是在“流程”部分处理完成之后，才执行响应行为。

请注意，`system.skipEntry();` 调用将指示 AssemblyLine 跳过响应行为，同样地，Server 方式连接器将不会回复。如果您更愿意完全跳过其余“流程”部分组件，可是还发送响应，请使用 `system.exitBranch("Flow");` 调用。

如果您未在 AssemblyLine 内直接创建连接器，那么为了在 AssemblyLine 中使用此连接器，请将其从 `<workspace>/Resources/Connectors` 中它所在的位置拖动到 AssemblyLine 的**供给**部分。

***Delta** 方式:*

设计 Delta 方式可以基于变化量操作代码通过提供对已连接系统的递增修改来简化数据更改的应用。

变化量操作代码由“迭代器变化量引擎”功能部件（迭代器的**变化量**选项卡）或 Change Detection 连接器（例如，IBM Security Directory Server、LDAP、Active Directory (AD)、RDBMS 和 Lotus Domino Changes 连接器）进行设置；或通过使用“轻量级目录交换格式”(LDIF) 或“目录服务标记语言”(DSML) 解析器解析该变化量信息来进行设置。

在 IBM Security Directory Integrator V6.1 之前的版本中，变化量引擎处理期间写入变化量存储（系统存储的一个功能部件）中的快照将立即落实。因此，即使处理“AL 流”部分失败，变化量引擎仍会将已更改的条目视为已处理。这种限制通过“连接器变化量”选项卡上的 *Commit* 参数来解决。该参数的设置控制变化量引擎对系统存储落实所捕获的入局数据快照的时间。

Delta 方式仅对于 LDAP 和 JDBC 连接器可用。

注：变化量方式的连接器必须与提供变化量信息的另一个连接器配对使用，否则变化量方式将没有可供使用的变化量操作码。

IBM Security Directory Integrator 中的“变化量”功能（请参阅第 189 页的第 7 章，『变化量』）旨在为同步解决方案提供帮助。您可以看到系统 Delta 功能分为两部分：*检测*和*应用*。

Delta 检测: IBM Security Directory Integrator 提供了大量更改或变化量、检测机制以及工具:

变化量引擎

这是对 Iterator 方式的连接器可用的功能。如果从迭代器的 Delta 选项卡上启用，Delta 引擎功能将使用“系统存储”来获取正在进行迭代处理的数据快照。然后在随后的运行中，经迭代处理的每个条目将会与快照数据库（变化量存储）进行比较以查看发生的更改。

Change Detection 连接器

这些组件利用已连接系统中的信息来检测更改，根据连接器的不同，它们用于 Iterator 或 Server 方式中。例如，Iterator 方式适用于许多 Change Detection 连接器，例如 LDAP、IBM Security Directory Server Changelog、RDBMS、Active Directory 以及 Notes/Domino 的 Change Detection 连接器。

Change Detection 连接器的设计旨在使它们行为一致，并对公共设置提供相同的参数标签。这些连接器是:

- IBM Security Directory Server Changelog
- AD Change Detection（Active Directory）

- Domino Change Detection
- Sun Directory Change Detection（openLDAP、SunOne、iPlanet 等）
- RDBMS Change Detection（DB2®、Oracle、SQL server 等）
- z/OS® LDAP Changelog

注: IBM Security Directory Integrator V7.2 及后续版本不支持 z/OS 操作系统。

请参阅参考中的“连接器”部分，以获取这些连接器的更多信息。

Delta 引擎功能自始至终报告属性的各个值的特定更改。在解析 LDIF 文件时，也可获得这种程度精细的更改检测。其他组件则只限于在添加、修改或删除整个条目的情况下才进行报告。

通过选择迭代器的“变化量”选项卡中的**允许重复变化量键**复选框，可指示允许重复的变化量键，在这些情况下，长时间运行的 AssemblyLine 需要多次处理相同的条目。这意味着当条目已更新时，可以处理使用更改日志或更改检测连接器、Delta 方式连接器和 Delta 方式存储的 AssemblyLine 的重复条目。

警告: 例如，可以具有带有大量 Changelog 和 Delta 方式连接器的 AssemblyLine。在这种情况下，如果 Delta 方式连接器指向作为 Changelog 连接器的同一底层系统，那么 Delta 操作可以再次触发 Changelog。由于没有方法可以区分新接收的更改和由 Delta 引擎触发的更改，所以应该认真考虑场景，以避免进入死循环。

由 Delta 引擎计算的变化量信息存储在“工作条目”对象中，并可根据使用的“更改检测”组件或功能存储为*属性级*乃至*属性值级*的*条目级操作代码*。

例如，设置一个启用 Delta 引擎功能的文件系统连接器。使这个连接器对一个可在文本编辑器中方便地进行编辑的简单 XML 文档执行迭代操作。例如:

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
    <Entry>
        <Telephone>
            <ValueTag>111-1111</ValueTag>
            <ValueTag>222-2222</ValueTag>
            <ValueTag>333-3333</ValueTag>
        </Telephone>
        <Birthdate>1958-12-24</Birthdate>
        <Title>Full-Time SDI Specialist</Title>
        <uid>jdoe</uid>
        <FullName>John Doe</FullName>
    </Entry>
</DocRoot>
```

请确保使用特殊的“映射全部”属性映射字符，即星号（*）。这是您在映射中唯一需要的属性，以确保返回的所有属性都映射至工作条目对象中。

现在通过以下代码添加脚本组件:

```
// Get the names of all Attributes in work as a String array
var attName = work.getAttributeNames();
// Print the Entry-level delta op code
task.logmsg(" Entry (" +
    work.getString( "FullName" ) + ") : " +
    work.getOperation() );
// Loop through all the Attributes in work
for (i = 0; i < attName.length; i++) {
```

```
 // Grab an Attribute and print the Attribute-level op code
att = work.getAttribute( attName[ i ] );
task.logmsg("    Att ( " + attName[i] + ") : " + att.getOperation() );
 // Now loop through all the Attribute's values and print their op codes
for (j = 0; j < att.size(); j++) {
 task.logmsg( "      Val (" +
                   att.getValue( j ) + ") : " +
                   att.getValueOperation( j ) );
}
}
```

第一次运行这个 AL 时，您的脚本组件代码将创建如下的日志输出:

```
12:46:31   Entry (John Doe) : add
12:46:31      Att ( Telephone) : replace
12:46:31        Val (111-1111) :
12:46:31        Val (222-2222) :
12:46:31        Val (333-3333) :
12:46:31      Att ( Birthdate) : replace
12:46:31        Val (1958-12-24) :
12:46:31      Att ( Title) : replace
12:46:31        Val (Full-Time SDI Specialist) :
12:46:31      Att ( uid) : replace
12:46:31        Val (jdoe) :
12:46:31      Att ( FullName) : replace
12:46:31        Val (John Doe) :
```

因为在先前的空“变化量存储”中找不到该条目，所以它将在条目级标记为 *new*。而且，它的每个属性都具有 *replace* 代码，这表示所有值都进行了更改（这很有意义，因为 Delta 表明这是新数据）。

对您的 XML 文件进行以下更改:

1. 将最后的 Telephone 数字值更改为 333-3334。
2. 删除 Birthdate。
3. 添加新的 Address 属性。

您的结果配置应该类似于以下内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
    <Entry>
        <Telephone>
            <ValueTag>111-1111</ValueTag>
            <ValueTag>222-2222</ValueTag>
            <ValueTag>333-3334</ValueTag>
        </Telephone>
        <Title>Full-Time SDI Specialist</Title>
        <uid>jdoe</uid>
        <FullName>John Doe</FullName>
        <Address>123 Willowby Lane</Address>
    </Entry>
</DocRoot>
```

再次运行 AL。这一次您的日志输出应该类似于这样:

```
13:53:22   Entry (John Doe) : modify
13:53:22      Att ( Telephone) : modify
13:53:22        Val (111-1111) : unchanged
13:53:22        Val (222-2222) : unchanged
13:53:22        Val (333-3334) : add
13:53:22        Val (333-3333) : delete
13:53:22      Att ( Birthdate) : delete
13:53:22        Val (1958-12-24) : delete
13:53:22      Att ( uid) : unchanged
```

```
13:53:22        Val (jdoe) : unchanged
13:53:22      Att ( Title) : unchanged
13:53:22        Val (Full-Time SDI Specialist) : unchanged
13:53:22      Att ( Address) : add
13:53:22        Val (123 Willowby Lane) : add
13:53:22      Att ( FullName) : unchanged
13:53:22        Val (John Doe) : unchanged
```

现在条目将标记为 *modify*，且属性反映了对其中每个条目所作的更改。可以看到 Birthdate 属性已标记为 *delete*，而 Address 标记为 *add*。这正是您对“输入映射“使用特殊的“映射全部”字符的原因。如果您只映射了该 XML 文档第一版中存在的属性，则当 Address 出现在输入中时，我们将无法检索到它。

特别应注意 Telephone 属性下标记为 *modify* 的最后两个值条目。对该属性的其中某个值的更改产生了两个“变化量”项，值为 *delete*，和值为 *add*。

要在先前版本的 IBM Security Directory Integrator 中构建数据同步 AssemblyLine，必须编写脚本以便处理流程控制。虽然您可以从您的更改组件或功能部件中接收 *adds*、*modifies* 和 *deletes*，但是连接器仅能设置成两种必需的输出方式中的一种：Update 或 Delete。因此，您要么具有两个指向同一目标系统的连接器，并且将脚本放入每个连接器的 Before Execute 挂钩中，以便当其操作码不匹配该组件方式时忽略条目；要么具有处于 *Passive* 状态的单个连接器（Update 或 Delete 方式），然后从您检查操作码的脚本代码控制其执行。这仍意味着即使您在已修改条目的情况下知道所更改的内容，Update 方式连接器仍然会先读入原始数据，然后再将更改回写到数据源。当您仅更改包含数千个值的多值组的相关属性中的单个值时，这可能造成不必要的网络或数据源流量。

进入连接器 *Delta* 方式。

Delta 应用（连接器 Delta 方式）： Delta 方式的设计旨在简化变化量信息的应用；即，通过多种方式进行实际更改。

首先，Delta 方式处理所有类型的变化量：添加、修改和删除。这将数据同步 AL 的数量减少为两个连接器：供给部分的一个“Delta 检测连接器”用来获取更改，而另一个 Delta 方式的“Delta 检测连接器”用于将这些更改应用到目标系统。

而且，Delta 方式将把 Delta 信息应用到目标系统本身所支持的最低级别。方法是：首先检查“连接器接口”以查看受数据源支持的递增修改的级别[3]。如果您使用的是 LDAP 目录，那么 Delta 方式将执行属性值的添加和删除。对于传统的 RDBMS（JDBC），删除一个列值再添加一个列值没有意义，所以对于这一类属性处理成值替换。

这可以通过支持该功能的那些数据源的 Delta 方式自动处理[4]。如果数据源提供了优化的调用来处理递增修改，并且连接器接口支持这些调用，那么 Delta 方式将使用这些调用。另一方面，如果连接的系统不提供“智能”变化量更新机制，Delta 方式将尽可能模拟这些机制，执行预更新查找（比如 Update 方式），更改计算以及对检测到的更改的后续应用。

3. 请注意，支持递增修改的仅有的连接器是 LDAP 和 JDBC 连接器，因为 LDAP 目录提供该功能

4. 而且，可以通过配置参数和挂钩代码控制这些内置行为。

链接条件:

“链接条件”用于告知处于 Update、Lookup 和 Delete 方式的连接器如何定义 AssemblyLine 中的数据属性和已连接系统中找到的数据属性之间的匹配。

在配置编辑器中通过**链接条件**选项卡访问“链接条件”，其中“连接条件”仅用于 Update、Lookup 和 Delete 连接器方式。

有两种类型的“链接条件”，**简单**和**高级**。

简单链接条件: 对于每个简单链接条件，指定连接器属性（在“连接器模式”中定义的那些属性）、要使用的运算符（例如，Contains、Equals 等）以及要在操作中使用的值。您使用的值可以直接输入，或可以引用工作条目（此时在 AssemblyLine 流程中可用）中的属性值。当连接器执行 Lookup 操作（对于 Lookup、Update 和 Delete 方式）时，它将“链接条件”转换为特定于数据源的调用，使您能够保持解决方案独立于底层技术。

如果想要使用工作条目中某个属性的值构建“链接条件”，只需在“链接条件”的值字段中使用该属性的名称，名字前面加上美圆符号（$）。因此，如果想要名为 *cn* 的属性与工作条目中某个名为 FullName 的属性相匹配，则“链接条件”应指定为:

```
cn EQUALS $FullName
```

如果想要直接查找某个特定的人，则用文字常量值设置“链接条件”:

```
cn EQUALS Joe Smith
```

注: 美圆符号（$）只匹配多值属性的第一个值。如果想要数据源中的某个属性与存储在工作条目属性中的多个值中任意一个值相匹配，则使用 @ 符号（@）。例如:

```
dn EQUALS @members
```

此示例尝试将已连接系统中的 dn 属性与名为 *members* 的工作条目中多值属性的任意一个值相匹配。

连接器可以有多个已定义的“链接条件”，并且这些“链接条件”通常连接在一起（使用布尔运算符 AND）以查找匹配。

但是，如果单击**匹配任何**，就只需匹配一个“链接条件”，这相当于 OR 操作。

请注意，要匹配的属性名称可以指定为“表达式”（请参阅第 31 页的『表达式』以获取详细信息）。“简单链接条件”的值字段可能使用的格式有:

文本字符串
映射至具有该值的常量。

$Name
与 `work.getString("Name")` 对应，即属性 *Name* 的第一个值。

@Name
与多值属性 *Name* 的其中一个值匹配。

IBM Security Directory Integrator 表达式
这里有详细的描述: 第 31 页的『表达式』。

高级链接条件: 您可以通过选中**构建具有定制脚本的条件**复选框来创建您自己的定制搜索条件。这会为您提供脚本编辑器以编写自己的“链接条件”表达式。并非所有连接器都支持“高级链接条件”，连接器文档状态会声明是否支持“高级链接条件”。请参阅 参考 中的“Connectors”章节。

您构建的搜索表达式必须符合底层系统预期的语法。要将您的搜索表达式传递至连接器，必须用字符串表达式填充 *ret.filter* 对象。

SQL 连接器的简单 JavaScript 示例为:

```
ret.filter = " ID LIKE '" + work.getString("Name") + "'";
```

该定制链接条件假定一个示例，其中数据源具有名为 *ID*（通常是列名）的属性，我们想要它与工作条目中的 *Name* 属性相匹配。

注:

1. SQL 表达式的第一部分 `Select * from Table Where` 由 IBM Security Directory Integrator 提供。
2. 已经添加了单引号，因为 `work.getString()` 返回字符串，而 SQL 语法要求字符串常量两边带有单引号。
3. 这里未使用带有 **$** 和 **@** 的特殊语法。

链接条件错误

使用“链接条件”时最常见的错误是:

```
ERROR> AssemblyLine x failed because
No criteria can be built from input (no link criteria specified)
```

当您在 Lookup 方式下找不到“链接条件”引用的属性，则会发生该错误。例如，使用以下的“链接条件”:

```
uid equals $w_uid
```

如果工作目录中不存在 *w_uid*，那么“链接条件”设置将失败。这可能由于没有从输入源将其读入（例如，不在输入映射中，或输入源中缺少该项）或已从脚本中的工作条目中将其除去。换句话说，函数调用 `work.getAttribute("w_uid")` 返回 NULL。

避免这种情况发生的一种方法是在 Lookup、Delete 或 Update 方式的连接器的 Before Execute 挂钩中编写代码，以便由于缺少属性而无法解析“链接条件”时跳过该操作。例如:

```
if (work.getAttribute("w_uid") == null)
  system.ignoreEntry();
```

您的业务规则可能需要其他处理（例如用 `skipEntry()` 调用代替 `ignoreEntry()` 调用），这会使 AssemblyLine 停止当前条目的处理并从新的迭代顶端开始处理。而 ignoreEntry() 函数只是跳过当前连接器并继续处理 AssemblyLine 的剩余部分。

函数

*函数*通常指的是函数组件（FC），除了不具有方式设置以外，它是与连接器非常类似的组件。但是，连接器为连接的系统提供了标准访问动词（Lookup、Delete、Update 等），另一方面，函数仅执行单个操作，例如，通过解析器推送数据、向另一条 AssemblyLine 分派作业或进行 Web service 调用。

概述

函数可以出现在 AL“流程”部分的任何位置。配置浏览器中的“函数”库文件夹可用于管理您的“函数组件”库。

与连接器类似，AssemblyLine 中的函数提供了一个“初始化”选项来确定该组件的启动时间。缺省情况下，函数在 AL 启动时进行初始化。

脚本组件

脚本组件（SC）是用户定义的 JavaScript 代码块，您可将其放置在 AssemblyLine 数据流列表中的任何位置，与连接器和函数组件一起，使其中的脚本代码在 AL 工作流中每次循环的这个时刻执行。

概述

与挂钩不同，“脚本组件”可以很容易地在 AssemblyLine 流程中随意移动，这使得“脚本组件”无论对于调试原型或实施您自己的流逻辑都是非常有用的工具。另外，与其他类型的 AL 组件不同，脚本没有任何预定义的行为或方式；您可以使用您的脚本实施行为和方式。脚本库可以存储在配置浏览器的脚本库文件夹中。

用途

例如，如果您只想调试 AssemblyLine 的某一部分，您可将以下代码放置在 SC 中，以限制和控制 AL 流。

```
task.dumpEntry( work );
system.skipEntry();
```

当放在 AssemblyLine 的流中合适的点时，此 SC 将显示工作对象的内容，方法是：将其写入到日志输出，然后跳过此循环的 AL 的余下部分。通过在组件列表中上移或下移 SC，您可以控制 AL 实际执行的量。如果将 `system.skipEntry()` 调用换成 `system.skipTo("ALComponentName")`，则可以直接将控制传递到特定的 AL 组件。

还可以使用 SC 来驱动其他组件。在执行目录或数据库同步时的一个典型情况是必须同时处理已更新信息和已删除信息。因为由内建 AL 工作流驱动的连接器一次只能运行于一个模式中（Update 或 Delete），所以您将需要用自己的代码对该逻辑进行一些扩展。一种方法是添加两个连接器，一个处于 Update 方式，另一个设为 Delete 方式，然后将代码放在每个连接器的 Before Execute 挂钩中，以告知这些连接器跳过它们不应该处理的更改操作。例如，在 Update 连接器的 Before Execute 挂钩中，您可以编写类似以下内容的代码：

```
// The LDAP change log contains an attribute called "changeType"
if (work.getString("changeType").equals("delete"))
 system.ignoreEntry();
```

这将使 Update 方式的连接器跳过已删除的“条目”。可以在您的 Delete 方式连接器的 Before Execute 挂钩中放置补充代码，跳过除了要删除的条目之外的任何其他条目。

但是，如果要同步多个目标的更新，这就需要每个数据源有两个连接器。另一种方法是只要一个通过脚本驱动的处于“被动的”状态的连接器。举个例子，假设您具有一个被称为 *synchIDS* 的被动 AL 连接器。接着，您可以添加带有以下代码的 SC 来运行该连接器：[5]

```
if (work.getString("changeType").equals("delete"))
 synchIDS.deleteEntry( work )
else
   synchIDS.update( work );
```

只要您明确地标出了您的 SC，指明它正在运行被动连接器，那么这个方法将产生更加短小的 AssemblyLine，更易于读取和维护。这是一个不得不在两种最佳实践中作出选择的示例：要么缩短 AL，要么使用内建逻辑与定制脚本。但是，对于本例的情况，实现易读性和简单性目标的最佳方式是编写少量的脚本。

当您希望测试逻辑和脚本代码时，脚本组件也派得上用场。只要在数据流列表中创建带有单个脚本组件的 AssemblyLine，将它置于您的代码中并运行它即可。

另请参阅

第 39 页的第 2 章，『IBM Security Directory Integrator 中的脚本编制』。

属性映射

属性映射是数据流入或流出 AssemblyLine 的通道。属性映射出现在连接器和函数中作为输入和输出映射，还可用作 AssemblyLine 中独立的组件。

概述

标题为第 1 页的『AssemblyLine』的部分开头的图以曲线箭头描述了三个属性映射：两个输入映射将数据引入到 AssemblyLine 中，一个输出映射将数据传递给连接器的高速缓存（*Conn* 条目）以便可以进行编写。

每个属性映射都包含在 Work 条目或 Conn 条目中创建属性的规则列表。映射规则指定两点：

1. 要在目标条目中创建（或覆盖）的属性的名称。对于输入映射和独立属性映射组件而言，这是 Work 条目，而对输出映射目标而言，则为 Conn 条目。
2. 用于使用一个或多个值填充属性的赋值。赋值可以是赋值脚本，例如：

   ```
   work.Title
   ```

   ，也可以是具有可选标记替换的文字文本（包括换行符）：

   ```
   <html>
      <header>
         <title>{work.Title}</title>
      </header>
      <body>
         <p>Look for the "Title" Attribute value in the title of this page</p>
      </body>
   </html>
   ```

属性映射是使用配置编辑器创建的；请参阅第 96 页的『属性映射和模式』。

5. 由于被动连接器不是由 AL 逻辑运行的，所以这些连接器出现在数据流列表的哪个位置无关紧要。

属性映射支持继承，不仅在映射级别，也针对单独的映射规则。注意，您可以将脚本拖动到属性映射中以设置继承的 JavaScript 映射规则。

属性映射还提供了称为『Null 行为』的功能，它用于控制处理所缺少数据的方式。

请参阅第 40 页的『内部数据模型：条目、属性和值』了解关于属性映射的更多信息。

Null 行为

有时，系统会尝试映射缺少的属性。例如，如果某个可选的电话号码不在输入数据源中，或输出映射中的某个属性已从工作条目中移除。其他时候，虽然属性存在，但是其没有值 – 如数据库表中的可空列。

不同的数据源以不同的方式处理缺少的值（空值、空字符串），而本部分中描述的功能提供了一种方式来定制如何处理缺失的属性或属性值。这种功能称为 *Null 行为*，有此功能，您可以定义什么是“空值”以及如何处理空值。

注：JDBC 连接器具有 `jdbcExposeNullValues` 参数设置，使您可以将空值映射到缺失的属性（请参阅 *参考* 中的“JDBC Connector”）。

Null 行为可以在许多级别指定：系统、配置、AssemblyLine、属性映射和属性。但是，由于 Null 行为是高度特定于数据源的，因此这使得在属性映射级别（例如，由连接器的输入/输出映射处理的所有属性）设置这个系统特性具有非常的意义。此处以更详细的方式描述了这些可能的级别：

系统级别

指定系统级别的 Null 行为是在 global.properties 文件中通过将 `rsadmin.attribute.nullBehavior` 和 `rsadmin.attribute.nullDefinition` 属性设置为稍后在本部分中列出的某个值而完成的。

配置级别

这会覆盖系统级别的 Null 行为，并且是通过将属性存储中的 `rsadmin.attribute.nullBehavior` 或 `rsadmin.attribute.nullDefinition` 属性设置为稍后在本部分中列出的以下某个值来进行配置的。

AssemblyLine 级别

AssemblyLine Null 行为是通过单击“AssemblyLine”工具栏中的**选项...**按钮，选择**AssemblyLine 设置**，然后在出现的对话框中单击**空值行为**来指定的。

属性映射级别

为映射中的所有属性定义 Null 行为是通过单击属性映射列表上的**更多...**按钮，然后选择 **Null 行为**来完成的。

属性级别

可以通过在属性映射中右键单击特定属性并选择 **Null 行为**来配置此属性的 Null 行为。已经为某属性进行了此定义时，之后会通过映射项中存在的蓝色项目符号来进行指示。

Null 行为支持用于定义什么是空值的五种不同设置，如下所示（请注意：每个设置的括号中的文本是用于设置系统级别 Null 行为定义的值，且通常在解决方案或全局属性存储中定义此文本）。每个设置都在圆括号中显示实际的特性值。请注意这些定义是以包含顺序列出的，因此第二种情况还包含第一种情况，而第三者情况包含前两种情况，等等：

属性缺失（AbsentAttribute）
缺少属性映射中引用为值源的属性。

属性无值（EmptyAttribute）
找到了在属性中用作值源的属性，但是没有值。还要检查先前的情况。

属性包含空字符串值（EmptyString）
属性已找到，但该属性只具有单个字符串值。

值（value）
属性包含指定的值。对于 AssemblyLine、属性映射和属性级别的空值定义，此值设置在“Null 行为”对话框的**值**字段中。如果需要，您在这里可以通过将值放置在不同的行来指定多个属性值。如果对系统和配置级别设置使用 `rsadmin.attribute.nullDefinition`，那么还必须设置 `rsadmin.attribute.nullDefinitionValue` 属性。

注: 已对 HTTP server 连接器进行了许多增强。基于 TCP 的组件（例如 HTTP server 连接器）的配置屏幕中有一个 switch，用于将 TCP 头作为属性值返回。如果清除了此标志，那么 TCP 头将存储为已返回的条目对象中的特性。

缺省行为（Default Behavior）
必须从更高级别继承空值定义。例如，属性从属性映射设置继承其空值定义，而属性映射设置转而从 AssemblyLine 进行继承。

注: 配置级别的 Null 行为可覆盖任何系统级别的设置。而且，在系统级别设置的缺省行为等同于指定**删除**，而在配置级别其等同于**值**。

“Null 行为”功能还使您能够在检测到空值时定义要采取的操作:

空字符串（empty string）
缺少的属性用具有空字符串值（""）的单个值来映射。

Null（null）
未用值映射缺少的属性，意味着 `att.getValue()` 调用返回 **null**。

删除（delete）
从映射删除该属性。

值（value）
用指定的值映射缺少的属性。对于 AssemblyLine、属性映射和属性级别的 Null 行为，值设置在“Null 行为”对话框的**值**编辑中。如果需要，您在这里可以通过将值放置在不同的行来指定多个属性值。如果您为系统和配置级别的设置使用 `rsadmin.attribute.nullBehavior`，则必须还要设置 `rsadmin.attribute.nullBehaviorValue` 属性。

缺省行为（Default Behavior）
必须从更高的级别继承 Null 行为。例如，属性级别从属性映射继承，而 AssemblyLine 级别从 AssemblyLine 设置继承。

注: 配置级别的 Null 行为可覆盖任何系统级别的设置。而且，在系统级别设置的缺省行为等同于指定**删除**，而在配置级别这等同于**值**。

分支组件

分支组件影响其他组件（如连接器、脚本、函数、属性映射和其他分支组件）在 AssemblyLine 流程中执行的顺序。

概述

分支组件有三种类型:

- 简单（称为分支）
- 循环（循环分支）
- Switche（共享同一表达式的分支）

分支可以出现在“流程”部分的任何位置，但是工作空间的“资源”部分中没有它们的库文件夹。

分支不必完成运行；使用相同的脚本调用可通过编程方式退出任何类型的分支: `system.exitBranch()`。关于更多信息，请参阅第 25 页的『退出分支（或循环或 AL 流）』。

三种分支组件的类型分别是:

分支

而每类分支用于为 AssemblyLine 处理定义备选路径，这种最简单的形式可以确定以下情况中的操作: “如果发生该情况，那么执行该操作。”可通过设置必须满足的条件而定义“发生情境”的含义；例如，比较数据值或检查某个操作的结果。如果条件为 True，则执行附加到该分支下面的组件。

您可以根据 IBM Security Directory Integrator 服务器中的以下任何数据定义条件: 属性值、参数设置、外部可访问属性以及通过 JavaScript 可用的任何信息（例如，磁盘的操作系统调用或内存使用情况）。根据**匹配任何**复选框的设置，多个条件可进行“与”或“或”操作。

这种最简单的分支组件格式也支持三种子类型设置，您可以选择 IF、ELSE-IF 和 ELSE。

IF 可以出现在 AssemblyLine 流程中的任何位置，如果条件为 True，那么 IF 分支将为处理提供可使用的备用路径。一旦执行了分支下的组件，则控制将传递给该分支*之后*的第一个组件。如果不希望该情况发生，则必须添加 ELSE 或 ELSE IF 分支，或通过脚本调用 `system.exitBranch()` 退出分支。

ELSE-IF

除了它只能紧跟在 IF 或 ELSE-IF 分支后面以外，其余与 IF 分支是相同的。

ELSE 只能紧跟在 IF 或 ELSE-IF 分支后，ELSE 分支没有条件。仅当没有前面的 IF 或 ELSE-IF 分支为 True 时，它的组件才会被处理。此外，对 ELSE 实例求值的结果始终为 true，因此循环中不对条件求值。

上面已提到，您可以通过脚本编制，使用 `system.exitBranch()` 提早退出分支。

循环

循环组件提供了在 AssemblyLine 中添加循环逻辑的功能。可以为三种操作方式配置循环: 基于条件、基于连接器或基于属性值:

有条件 就像对简单分支一样，可以定义控制循环行为的条件。只要满足条件循环就会继续，直到条件不满足才会停止。循环的详细信息窗口与前一节中描述的简单分支的窗口相同。

连接器 该方法使您可以在 Iterator 或 Lookup 方式下设置连接器，并且通过循环流对每个返回的条目进行循环。该类循环的详细信息窗口包含配置连接器、连接和发现属性以及设置输入映射所必需的“连接器”选项卡。

注意：您具有名为**初始选项**的参数，利用它可以指导 AL 执行以下操作：

- **不执行任何操作**表示不会在 AL 循环之间做任何方式的连接器准备工作。
- **初始化并选择/查找**使得在每个 AL 循环期间重新初始化连接器。
- **仅选择/查找**，保持连接器初始化，但是要根据方式设置重新执行迭代器选择或查找。

另请注意，有一个**连接器参数**选项卡的功能类似于“输出映射”，在此您可以选择要从 **work** 属性值设置的连接器参数。

这带给我们一个论题，即使用迭代器的循环与基于 Lookup 方式的循环有何不同。两个选项都执行*搜索*，该搜索创建为循环返回的结果集。对于 Iterator 方式，结果集专门由该组件的参数设置控制。另一方面，对于 Lookup 方式，则使用“链接条件”定义搜索或匹配规则。由于不需要对 On No Match 或 On Multiple Found 之类的挂钩编码，因此这是执行那些可能不会始终返回一个（且仅有一个）匹配条目的搜索的首选方法。

属性值 通过选择工作条目中任何可用的属性，将针对每个属性值执行“循环”流量。每个值都会传递到循环的第二个参数中指定的新工作条目属性中。该选项使您能够很容易处理多值属性，例如组成员列表或电子邮件。

您可以通过脚本编制，使用 `system.exitBranch()` 提早退出循环。

Switch

与在分支条件和循环条件中使用的表达式不同，Switch 表达式可以生成的值不仅仅限于 `true` 或 `false`。例如，当从另一条 AssemblyLine 或另一个进程调用该 AL 时，您可以打开属性的值。在 Switch 组件中，对想要处理的 Switch 表达式的每个常量值添加一种情况。例如：如果设置 Switch 来使用工作条目中的变化量操作码，则您的 Case 情况将会是类似“add”、“delete”和“modify”的值。

在 AL Switch-Case 构造中，多种情况可以同时成立。IBM Security Directory Integrator 会检查每种情况，就好像这是一系列标准的 IF 分支序列一样。以下示例显示多种情况是如何工作的：

```
work.setAttribute("test","abc");

Switch work.test
  Case startsWith("a"): this is true
  Case contains ("bc"): this is true
  Case length=3: this is true
```

三个 Switch work.test 表达式都为 True 将会触发 Switch 执行。

您可以通过脚本编制，使用 `system.exitBranch();` 提早退出 Switch-Case。

退出分支（或循环或 AL 流）

如果要退出分支、循环或 Switch，甚至如 AL 流程部分之类的内置分支，那么在可以编写脚本（例如: 挂钩或甚至脚本组件）的位置使用 `system.exitBranch()` 方法。不带参数（或者带空字符串）调用 `system.exitBranch()` 将使包含的分支退出，而流程继续处理分支之后的第一个组件。

也可以给此方法提供包含以下任何一种内容的字符串参数:

保留的关键字之一: Branch、Loop、Flow、Cycle 或 AssemblyLine（不区分大小写）
这将中断这一类型的第一个分支，并顺着 AssemblyLine 向上回溯。因此，如果您的脚本代码位于循环内的分支中，并且您执行了调用 `system.exitBranch("Loop")`，则会退出分支以及包含该分支的循环。使用保留字 `Flow` 将使流程退出 AssemblyLine 的“流程”部分，从而在有 Server 方式连接器的情况下继续执行响应行为，或继续执行到下个条目中将要读取的活动迭代器，或者继续进行 AL 关闭（Epilogs，...）。`Cycle` 关键字将控制传递到当前 AL 循环的结尾，并且不调用 Server 方式连接器中的响应行为，而 `AssemblyLine` 关键字将使 AL 停止并关闭。

`system.exitBranch()` 调用中使用的所有其他值将导致从具有指定名称的分支/循环中跳出。因此，调用 `system.exitBranch("IF_LookupOk")` 将在包含的分支或循环调用“IF_LookupOk”之后发送流。请注意，与将控制传递到任何指定 AL 组件的 `system.skipTo()` 不同，`system.exitBranch()` 将在指定的循环/分支后继续进行处理。

分支或循环的名称（区分大小写）
如果传入分支或循环（您的脚本调用嵌套在其中）的名称，那么控制将传递到 AL 中该分支或循环之后的组件。如果未发现具有此名称的分支或循环（从调用点开始回溯），那么会产生错误。

循环组件中还有继续功能。以下方法可用于系统对象中:

```
system.continueLoop();
system.continueLoop(name);
```

其中 *name* 是区分大小写的字符串，表示循环名称。在提供了循环名称的情况下，会将程序流转移到具有该名称的循环组件中。

解析器

解析器可与字节流组件（例如，文件系统连接器）结合使用来解释或生成正读取或写入的内容的结构。

请注意，当您试图解析的字节流与所选择的解析器不一致时，那么您将得到 `sun.io.MalformedInputException`。例如，当使用**输入映射**选项卡浏览文件时，会显示此错误消息。

配置编辑器提供了两个位置供您选择解析器:

1. 在字节流连接器的**解析器**选项卡中。
2. 在您自己的脚本中；例如，挂钩和脚本组件。

有关单个解析器的更多信息，请参阅 *参考* 中的“Parsers”。

字符编码转换

Java2 将 Unicode 用作其内部字符编码。Unicode 是双字节字符集。当您在 AssemblyLine 和连接器中使用字符串和字符时，始终假定它们是 Unicode。大多数连接器提供了一些字符编码转换的方法。从本地系统上的文本文件中读取时，Java2 已建立了一个缺省字符编码转换，这依赖于正在运行的平台。

IBM Security Directory Integrator 服务器具有 **-n** 命令行选项，在写入新的配置文件时该选项可用于指定要使用的字符集；该服务器还会在文件中嵌入该字符集指示符，因此，在稍后读回该文件时能够对其进行正确解析。

但是，您经常会从/到用不同字符编码来对信息进行编码的文本文件中读/写数据。例如，需要解析器的连接器通常在解析器配置中接受一个**字符集**参数。该参数必须设置为 IANA 字符集注册表（http://www.iana.org/assignments/character-sets）所指定的某个已接受的转换表。

某些文件，如果是用 UTF-8、UTF-16 或 UTF-32 编码的，则可能在文件的开头包含字节顺序标记（Byte Order Marker，BOM）。BOM 是字符 `0xFEFF` 的编码。它可用作所用编码的特征符。但是，IBM Security Directory Integrator 文件连接器无法识别 BOM。

如果您尝试读取带有 BOM 的文件，则应将此代码添加至连接器的 Before Selection 挂钩之类的位置:

```
var bom = thisConnector.connector.getParser().getReader().read(); // skip the BOM = 65279
```

这段代码将读取并跳过 BOM（假设您已经为该解析器指定了正确的字符集）。

对于 HTTP 协议，必须要注意某些事项；请参阅 *参考* 中对 HTTP 解析器的描述中有关字符集编码的部分以了解更多详细信息。

访问您自己的 Java 类

只要您自己的定制 Java 类是*公共*类和方法，就可以从内部的 IBM Security Directory Integrator 框架中进行访问。这些库必须打包到 .jar 或 .zip 文件中，然后放到 `TDI_install/jars` 目录中，最好放到您自己的子目录中。您还可以使用 CLASSPATH 环境变量或 Java 运行时环境扩展文件夹，但不建议使用这两种方法。只有当装入程序在装入您自己的类之前碰巧已装入了类，这些方法才使您可以从自己的类中调用类。

如果正在从配置编辑器运行服务器，必须先重新启动配置编辑器，然后才能检测 `TDI_install/jars` 目录和子目录中的新类。

当将 .jar 文件放入 jars 子目录之后，您可以创建该类的实例，以在 IBM Security Directory Integrator 中引用该实例。请注意，Java 函数组件允许您打开 .jar 文件，并浏览其中包含的对象以及它们的方法。一旦您选择了要调用的函数，FC 将为您准备 Input 和 Output 模式，以匹配 Java 函数所需的参数。

关于从脚本调用 Java 类的更多信息，请参阅第 65 页的『实例化 Java 类』。

使用配置编辑器实例化类

使用配置编辑器中**解决方案记录和设置**窗口的 Java Libraries 文件夹可以声明类。只有当您的类具有不需要参数的构造函数（通常但不总是缺省构造函数）时，这才有效。

添加类对象时，请单击**添加...**并指定两个参数: 脚本对象名（即作为 Java 类实例的脚本编制变量的名称）和 Java 类名称。例如，您可以有一个“脚本对象名称”*mycls*，而 Java 类可能是 *my.java.classname*。*mycls* 对象对于 Global Prologs 执行之前定义的任何 AssemblyLine 均可用。

注: 请注意，这将导致要为每个 AssemblyLine 的运行实例化您的对象。如果不希望这样，并且如果您更喜欢按照需求实例化，则请参阅下一节。

类的运行时实例化

如果您想要在指定的执行点或对于没有缺省构造函数的类进行实例化，则需要在运行时期间实例化类。例如:

```
crytoLib = new com.acme.myCryptoLib();
```

AssemblyLine 流程和挂钩

AssemblyLine 提供内置的自动化行为，它可以帮助您快速构建和部署数据流。这种自动化行为在 *参考* 中的流程图内有详细描述。此外，连接器和函数具有其自己的行为，这些也显示在流程图中。请注意连接器行为取决于方式设置。

这些内置逻辑流中存在大量*停留点*，在停留点中您可以添加自己的脚本化逻辑以扩展或完全覆盖内置行为。这些停留点称为挂钩，并可用于在所有连接器和函数以及 AssemblyLine 自身的挂钩选项卡下进行定制。

根据特定挂钩能否适用于正在运行的 AssemblyLine，您可以启用或禁用挂钩。禁用挂钩时，请不要中断它在所属连接器中的继承。

当启动 AssemblyLine 时，它将经过三个阶段: 启动、数据流和关闭。在启动期间，Prolog 挂钩必须先用于重新配置组件，然后才能进行初始化。在“数据流”阶段，每个送入 AssemblyLine 中的工作条目将会传递到“流程”组件进行处理。

最后，在关闭阶段，Epilog 挂钩可用于执行作业结束工作，例如，检查并报告错误状态，或存储状态数据以供下次 AL 启动时使用。

启动阶段

此时，将指示服务器装入并运行 AssemblyLine。服务器使用存储在配置文件中的蓝图来设置 AL。如果 TaskCallBlock（TCB）已经传递到 AssemblyLine，那么将会评估（这会产生 AL 组件参数变化）其内容。此时，Prolog 挂钩流将启动。

全局 Prolog

首先，如果定义了任何“全局 Prolog”，那么将对其进行评估。全局 Prolog 是位于项目的 Resources 文件夹中的脚本，这些脚本已包含在 AssemblyLine 中（在**解决方案记录和设置**窗口中）。通常在 AssemblyLine 的“AssemblyLine 设置”窗口中选择要在 AL 启动时运行的脚本来完成评估。完成了所有“全局 Prolog”后，将调用 AL Prolog 挂钩。

AL Prolog 挂钩（初始化前）

首先调用“AssemblyLine Prolog - 初始化前”挂钩。之后，配置成“在启动时”初始化的所有连接器和函数会经历其整个初始化阶段，在该阶段中也调用其 Prolog 挂钩，如下一点中所示。

连接器/函数初始化

对于每个具有“启动时”初始化设置的连接器和函数都会执行此初始化序列。这些连接器和函数将按其在 AssemblyLine 中定义的顺序依次启动。对于每个连接器或函数，序列流如下所示:

1. 调用了组件的 **Prolog - 初始化前**挂钩。
2. 启动组件；例如: 连接到其底层数据源、目标或 API。
3. 对于 Iterator 方式的连接器，处理 **Prolog - 选择前**挂钩，并且连接器执行条目选择；这将触发特定于数据源的调用，例如，执行 SQL SELECT 或 LDAP 搜索。
4. 对于迭代器，评估 **Prolog - 选择后**挂钩。
5. 调用 **Prolog - 初始化后**挂钩，以结束序列。

如果初始化某个连接器失败，那么 AssemblyLine 流程传递到可以处理此错误的 **Prolog - 错误**挂钩。

如果在设置或数据访问期间有错误发生，则重连接功能部件允许您配置连接器以自动尝试重新建立它的连接。这些设置在连接器的**连接失败**选项卡中可以找到。

注: 在该阶段评估脚本连接器（即使用 JavaScript 实施的连接器），从而注册所需的连接器函数并执行初始化代码。

AL Prolog 挂钩（初始化后）

将执行 **AssemblyLine Prolog - 初始化后**挂钩。该挂钩完成意味着启动阶段的结束和数据流阶段的开始。

数据流阶段

AssemblyLine 循环启动挂钩

在每个循环开始时调用该挂钩，然后调用“供给”或“流程”组件。

AssemblyLine 循环

控制将传递到流中第一个组件，通常是“供给”部分中 Server 或 Iterator 方式的连接器。

如果 AssemblyLine 中有一个或多个迭代器，那么第一个迭代器将通过从结果集检索下一个条目并将属性映射到工作条目中来启动循环。结果工作条目将传递到“流程”部分的组件中，位于列表的开头（如 CE 中所示）。

对于服务器方式的连接器，将启动侦听器进程以等待入局的客户机连接。当检测到连接请求时，连接器将克隆其自身并接受连接，然后将其自身转换成 Iterator 方式，以便从客户机将数据送入到“流程”部分以进行处理。无论哪种方式，您都可以获得迭代器将工作条目推送到“流程”组件。（同时，原始服务器方式连接器等待其他入局连接请求。）

如果您的 AssemblyLine 既不具有 Iterator 方式连接器也不具有 Server 方式连接器，那么它是单触发式 AssemblyLine，通常用于处理由另一个正在调用的进程供给的初始工作条目（IWE）。

循环的结束

执行了最后一个流程组件时，可能会发生以下三种情况之一:

- 如果当前的工作条目来自迭代器，那么会将控制传递回迭代器以从其源获取下一条目。
- 在服务器方式连接器的情况下，将对客户机作出回复。
- 对于以手动循环方式调用的 AL，会将线程传递回调用程序，以便可以访问结果。

此时没有特定的挂钩，尽管可以在末尾插入脚本将它添加到 AssemblyLine 中。

数据的结束

数据结束是 Iterator 方式的挂钩，当到达输入数据集结尾时将调用它。此时，控制会传递到下一个供给连接器，或者 AssemblyLine 将进入“关闭”阶段。

关闭阶段

此时，AL 处理可能正常完成，也可能由于错误而异常终止。

AssemblyLine Epilog - 关闭前挂钩

系统会处理称作 **Epilog - 关闭前**的 AssemblyLine 挂钩。

连接器/函数关闭流

每个连接器和函数的 Epilog 挂钩将按其出现在配置编辑器中的顺序进行调用:

1. **关闭前**挂钩。
2. 执行关闭操作；例如，关闭连接或释放 API 回调。
3. **关闭后**挂钩。

AssemblyLine Epilog - 关闭后挂钩

最后，AssemblyLine **Epilog - 关闭后**挂钩运行。

服务器方式连接器设置

服务器方式的连接器启动时，它进入事件侦听方式。一旦事件被接收，它将克隆 AL 并继续等待更多的事件。同时在克隆过程中，连接器将其自身转换为 Iterator 方式并将控制传递到供给列表中的下个组件。该过程允许您具有多个活动的服务器方式连接器并同时将数据送入流中，例如让处于服务器方式的若干 HTTP 服务器连接器侦听不同的端口，但送入相同的 AL。虽然服务器方式连接器是 AssemblyLine 配置的一部分，但是它们作为单独的进程（线程）运行。

对于这一方式下的连接器，有一组额外的挂钩会得到评估。特定于服务器方式功能且用于处理入站连接的挂钩包括:

接受前的连接

这个挂钩在连接器进入侦听方式之前调用。

接受后的连接

一旦接收了连接，即调用这个挂钩。请注意，此时没有数据可用。为了检查传入的事件信息，请使用迭代器挂钩，例如 **After GetNext** 或 **GetNext Successful**。

接受连接时出错

如果任何一个服务器方式挂钩出错，或者在事件侦听期间收到来自数据源的错误，那么将执行这个挂钩。

- 如先前所提到的，如果有多个连接器处于 Iterator 方式（请参阅 第 6 页的『AssemblyLine 中的多个迭代器』），那么这些连接器按照其在配置中的显示顺序

（由顶部至底部）在堆栈中排列。例如，如果您有 **a** 和 **b** 两个迭代器，则调用 **a** 直到它不再返回条目，然后 AssemblyLine 转换到 **b**。

- 如果没有连接器处于 Iterator 方式，并且在 AssemblyLine 启动时，没有向该 AssemblyLine 提供初始工作条目 (IWE)（例如，通过来自另一个 AssemblyLine 的调用），并且没有在 AssemblyLine 或连接器 Prolog 挂钩中创建工作条目，那么该 AssemblyLine 仍然执行一次。

最后，有一个**关闭请求**挂钩，您可以对其编写代码，以便可以在外部请求时正确关闭在 AssemblyLine（而非使 AssemblyLine 未崩溃）。

可以从 *system* 对象使用特殊功能来跳过或重试当前的工作条目以及跳过连接器等。请参阅『控制 AssemblyLine 的流程』以获取更多详细信息。

处理严重错误的终止和清除

有多种方法可对 IBM Security Directory Integrator 内部错误以及 IBM Security Directory Integrator 连接器、解析器、函数组件中出现的错误进行检测和处理。这些方法包括:

- 错误挂钩，可在其中编写 JavaScript 代码以处理错误。IBM Security Directory Integrator 用户可访问此方法。另请参阅 『控制 AssemblyLine 的流程』。
- Java try-catch-finally 块，其确保小故障不会中断服务器，以及适当处理了所有错误。在核心 IBM Security Directory Integrator 服务器类中，此类块已经到位。

新的 JVM 关闭挂钩功能部件提高了服务器的可靠性。Java 关闭挂钩允许部分代码在按下 Control-C 键后，或当 JVM 因为某个其他原因（甚至 System.exit）关闭时执行某种处理。

JVM 关闭时，您可以指定要启动的外部程序。该外部程序可以从 JVM 关闭挂钩中启动。将使用 `global.properties` 或 `solution.properties` 文件中的可选属性来配置此外部程序:

```
jvm.shutdown.hook=<external application executable>
```

Shell 脚本和批处理文件也可以指定为该特性的值。

当调用 JVM 关闭挂钩时，将无法阻止 JVM 终止。但是，执行外部程序时，可以执行可定制的操作: 例如，发送 IBM Security Directory Integrator 服务器已终止的消息，执行清除操作，甚至在需要时重新启动新的服务器。

控制 AssemblyLine 的流程

挂钩是 IBM Security Directory Integrator 的内置自动行为中的可编程停留点，在其中您可施加自己的逻辑。

在 AssemblyLine、连接器和函数组件中均可以找到挂钩。例如，如果您想要完全跳过或重新启动 AssemblyLine 的各部分，通常可以从连接器中的挂钩执行此操作:

注: 下面的构造也可用来退出分支组件或循环组件。

system.ignoreEntry()

忽略当前连接器并继续处理下一个连接器中的现有数据条目。

system.skipEntry()

完全跳过（删除）条目（终止当前循环），将控制权归还给 AssemblyLine 的开始并从当前 Iterator 中获取下个条目。

system.exitFlow()

放弃当前条目的进一步处理，执行循环结束逻辑；例如，保存迭代器状态密钥（如果已为此配置了连接器），将控制返回到 AssemblyLine 的开始并从当前迭代器获取下个条目。

system.restartEntry()

从 AssemblyLine 的开头重新启动，强制当前迭代器复用当前条目。

system.skipTo(字符串名称)

跳过指定的连接器。

system.abortAssemblyLine(字符串原因)

由于指定的错误消息，异常终止整个 AssemblyLine。

注：如果在**错误挂钩**中放置任何代码并且不终止当前的 AssemblyLine 或 EventHandler，则处理会继续，而不管您如何到达错误挂钩。这意味着在脚本中甚至忽略语法错误。因此如果想要知道导致错误的原因，请务必检查 *error* 对象。

在先前的列表中描述的方法可以看作是 goto 语句，因此在该挂钩中不会再运行其他任何代码。例如：

```
system.skipEntry(); // Causes the flow to change
// This next line is never executed.
task.logmsg("This will never be reached");
```

注：缺少的错误挂钩与空的错误挂钩之间存在差异 – 即使并非总是能轻松地在配置编辑器中发现此差异。*空的*错误挂钩使系统复位导致调用挂钩的错误条件，之后服务器继续处理；而*缺少*或未定义挂钩将导致系统执行缺省错误处理（通常是异常中止 AssemblyLine）。

表达式

IBM Security Directory Integrator 提供了与 V6 兼容的表达式功能，您可以使用该功能在运行时计算参数和其他设置，从而动态配置您的解决方案。此功能将对先前版本中的特性处理功能进行扩展。

除了支持简单外部属性引用（完全兼容较早版本）外，表达式还会在 AL 或组件初始化和执行期间操作 AssemblyLine 和组件配置设置时提供更多功能。表达式还可用于属性映射以及条件和链接条件，从而减少先前动态构建配置解决方案所需的大部分脚本编制。IBM Security Directory Integrator 提供了便于构建这些表达式的表达式编辑器。

表达式功能构建在标准 Java `java.text.MessageFormat` 类提供的服务顶部。MessageFormat 类提供强大的替换和格式化功能。以下是概述此类及其功能的联机页面的链接：http://docs.oracle.com/javase/1.6.0/docs/api/java/text/MessageFormat.html。

注：基于本部分所示表达式的 MessageFormat 是 IBM Security Directory Integrator V.6 中参数替换的基础；在 V7 中，最佳实践将改用高级 (JavaScript) 表达式。

虽然某些对象的可用性取决于运行时状态（例如，是否定义了 *conn*、*current* 或 *error*条目），IBM Security Directory Integrator 除上述类中描述的功能之外，还提供了若干可

用于表达式的运行时对象。表达式语法提供用于访问这些对象中的信息（例如指定的条目对象中的属性或组件的特定参数）的简化表示法。

表 2. 脚本对象、其用法和可用性

IBM Security Directory Integrator 参考	值	可用性

work.*attrname[.index]*	当前的 AssemblyLine 中的 *work* 条目。
	
	可选 *index* 指的是属性的第 *n* 个值。否则使用第一个值。
	
	以下是高级属性映射:
	
	`ret.value = work.getString ("givenName") + " " + work.getString("sn");`
	
	可以只表示为:
	
	`{work.givenName} {work.sn}`
		AssemblyLine

conn.*attrname[.index]*	当前 AssemblyLine 中的 *conn* 条目。 可选 *index* 指的是属性的第 *n* 个值。否则使用第一个值。	属性映射期间的 AssemblyLine
current.*attrname[.index]*	当前的 AssemblyLine 中的 *current* 条目 可选 *index* 指的是属性的第 *n* 个值。否则使用第一个值。	属性映射以进行修改期间的 AssemblyLine
config.*param*	当前组件 AL 的配置对象。此外，如果 *config* 用在连接器、解析器或函数的参数中，则它指的是该组件*接口*的配置对象（例如，JDBC 连接器或 XML 解析器）。 *param* 是参数本身的名称，就如同您对 getParam() 或 setParam() 进行调用一样。例如，对于 JDBC 连接器，您可以创建以下引用: `{config.jdbcSource]`	AssemblyLine 事件处理程序 连接器 解析器 函数组件

alcomponent.*name.param*	指定 AssemblyLine 组件的组件接口参数值。
	
	name 是 AssemblyLine 组件的名称
	
	param 是 *name* 对象的参数名
	
	因此，以下表达式:
	
	`{alcomponent.DB2conn.jdbcSource}`
	
	等同于以下用脚本编制的调用:
	
	`DB2conn.connector.getParam ("jdbcSource");`
		AssemblyLine

表 2. 脚本对象、其用法和可用性 (续)

IBM Security Directory Integrator 参考	**值**	**可用性**
property[:storename].name property[:storename/bidi].name	*TDI-Properties* 引用。 可选 storename 以特定的特性存储为目标。如果未指定 storename，则使用缺省存储。 *name* 是属性名 *bidi*（如果存在）将设置参数值以将调用转发给引用的属性存储。当 *bidi* 存在时，不允许使用其他替换模式或文本。	始终
JavaScript<<EOF script code ... // Must contain "return" EOF **注**: v.6 语法: 在表达式编辑器中使用高级（JavaScipt）选项	*嵌入式脚本代码用来生成表达式的值。此脚本必须返回值。* 此处所用的"EOF"文本是终止 JavaScript 片段的任意字符串。在遇到带有 EOF 字符串的（或未设置 EOF 标志）的单行之前，一直收集 JavaScript - 请参阅以下注释。 请注意嵌入式 JavaScript 是使用 AssemblyLine 的脚本引擎实例评估的，因此您有权访问所有变量（除了用于编制脚本的变量外）。 **注**: 以下是一个用于添加对不支持多行的输入字段（例如链接条件或映射中的属性名称）有效的 JavaScript 的简化格式，因此这种格式不需要 EOF 行: `{JavaScript return work.givenName + " " + work.surName}`	始终

表达式中的嵌入式 JavaScript 具有对 AssemblyLine 的脚本引擎的访问权。因此，可以访问在 AssemblyLine 中的其他地方定义的脚本变量。请注意，如果您引用的变量或对象没有被专门列在本部分显示的表中，那么表达式评估程序将以 AL 的脚本引擎来进行检查以查看该处是否定义了这样的变量或对象。

组件参数中的表达式

当用于组件参数时，特别要注意以下对象:

表 3. 表达式中可用的特殊对象

对象	**值**
配置	组件的接口配置对象。
mc	配置实例的 MetamergeConfig 对象（`config.getMetamergeConfig()`）。
work	AssemblyLine 的工作条目。
task	AssemblyLine 对象。

例如，使用表名称参数设置为"Accounts"的 JDBC 连接器。然后可以单击 **SQL Select** 参数标签或字段旁边的**打开参数值对话框**按钮，选择**具有替换的文本**，然后将其输入对话框底部的大文本字段中:

```
select * from {config.jdbcTable}
```

这将采用“表名称”参数，并创建以下 SQL SELECT 语句:

```
select * from Accounts
```

或者您可以获取更高级的表达式，并尝试对 SQL Select 参数使用:

```
SELECT {JavaScript<<EOF

 var str = new Array();
 str[0] = "A";
 str[1] = "B";
 return str.join(",");
EOF

} FROM {property:mystore.tablename} WHERE A = '{work.uniqueID}'
```

嵌入式 JavaScript 将返回值“A,B”，该值将用来完成余下的表达式。如果称为 *mystore* 的属性存储的 *tablename* 属性设置为“Accounts”，且工作条目中将有一个 *uniqueID* 属性（值为“42”），那么最终结果为:

```
SELECT A,B FROM Accounts WHERE A = '42'
```

此求值结果不显示在 CE 中。只输入花括号将不会对参数值完成表达式求值。而在将表达式绑定到参数时，您有两个选项:

1. 按字段旁的**打开参数值对话框**按钮（或单击“参数”标签 ）并选择**具有替换的文本**以在参数输入字段中打开“表达式”对话框。可以在此对话框中的大文本字段中输入表达式。单击**确定**以输入表达式。
2. 将特殊的前同步信号 @SUBSTITUTE 手动输入参数输入字段中，后面紧跟表达式。例如:

   ```
   @SUBSTITUTEhttp://{property.myProperties:HTTP.Host}/
   ```

   注:

   - 建议不要使用最后一个方法（直接输入表达式）；请使用表达式编辑器。
   - 如果对“文件连接器”的文件路径的值选择了 **Text w/substitution**，并且输入了 `{work.fullPath}`，那么将发生以下错误: “CTGDIC114E 参数‘文件路径’是必需参数”。此结果是预期结果，因为假定配置编辑器显示在文本中应用您指定的替换内容的结果。在这种情况下，没有 `work` 对象，因为 `work` 对象仅在处于运行状态的 AssemblyLine 中定义。因此，该结果为空字符串。对于此参数，空字符串是错误，因此将显示一条错误消息。尽管如此，AssemblyLine 处于运行状态时，可能会存在 `work` 对象，并且评估将根据需要生成参数字符串。

链接条件中的表达式

链接条件中的表达式提供相似的预定义对象列表。请再次注意，您还有权访问当前在组装流水线的脚本引擎中定义的所有其他对象或变量。

表 4. 用于链接条件中表达式的预定义对象

对象	重要性
配置	组件的接口配置对象
mc	配置实例的 MetamergeConfig 对象（`config.getMetamergeConfig()`）
work	AssemblyLine 的工作条目

表 4. 用于链接条件中表达式的预定义对象 (续)

对象	重要性
task	组件本身或指定的组件
alcomponent	连接器或函数组件

例如，您要为连接器设置链接条件以在运行时确定要在匹配过程中使用的属性。除了工作条目中的标准数据属性外，还有一个带有字符串值“uid”的 *matchAtt* 属性。在此情况下，链接条件中将使用以下表达式:

```
{work.matchAtt}  EQUALS {work.uid}
```

等同于:

```
uid  EQUALS $uid
```

分支、循环和条件判断中的表达式

此处的表达式对象列表类似于链接条件的表达式:

表 5. 用于分支组件中表达式的预定义对象

对象	重要性
配置	组件的接口配置对象
mc	配置实例的 MetamergeConfig 对象（`config.getMetamergeConfig()`）
work	AssemblyLine 的工作条目
task	AssemblyLine
alcomponent	连接器或函数组件

可以将表达式用于属性名称和条件的操作数。还可以使用表达式配置条件判断组件。

使用表达式进行脚本编制

还可以直接从 JavaScript 代码使用表达式。以下是使用新的 ParameterSubstitution 类来构建表达式的示例:

```
var ps = new com.ibm.di.util.ParameterSubstitution("{work.FullName} -> {work.uid}");

map = new java.util.HashMap();

map.put("mc", main.getMetamergeConfig());
map.put("work", work);

task.logmsg(ps.substitute(map));
```

在测试 AssemblyLine 中运行多个迭代时，从 JavaScript 代码生成的表达式发布以下日志消息:

```
14:35:29  Patty S Duggan -> duggan
14:35:29  Nicholas P Butler -> butler
14:35:29  Henri T Deutch -> deutch
14:35:29  Ivan L Rodriguez -> rodriguez
14:35:29  Akhbar S Kahn -> sahmad
14:35:29  Manoj M Gupta -> gupta
```

条目对象

了解 IBM Security Directory Integrator 的一个基础是要知道在系统中如何存储和传输数据。这是通过使用称为*条目*的对象来实现的。条目对象可看作是“Java 存储区”，可以包含任何数量的属性：无、一个或多个。

属性也是 IBM Security Directory Integrator 中类似于存储区的对象。每个属性可以包含零个或多个*值*，这些值是从已连接系统中读取和写入到其中的实际数据值。属性值也是 Java 对象；其可以为字符串、整数和时间戳记；无论是什么值，都需要与该数据值的本机类型匹配。单个属性可以容易地保留不同类型的值。工作，单个属性的值在大多数数据源中往往倾向于同一类型。

虽然此*条目属性值*范例与轻量级目录访问协议（LDAP）目录条目的概念准确匹配，但这也是数据库中的行在 IBM Security Directory Integrator 中的表示方式，如同在网络上接收的文件、IBM Lotus Notes 文档和 HTTP 页面中的记录一样。来自 IBM Security Directory Integrator 使用的任何源的所有数据都作为具有属性及其值的条目对象而进行内部存储。

与 IBM Security Directory Integrator 的较早版本相反，从 v7.0 起，分层的条目对象就在 AssemblyLine 中受到支持，并受到属于 AssemblyLine 的某些组件的支持。扩展了条目对象以提供若干方便的方法来处理分层数据，虽然在缺省情况下，这是隐藏的且仅在您将其显式启用或将其与要求分层功能的组件一起使用的情况下才开始予以使用。其还实施 org.w3c.dom.Document，这将使其成为层次结构中的顶级节点。与此相关的更多信息，请参阅第 42 页的『使用分层 Entry 对象』。

有少量由 IBM Security Directory Integrator 创建和维护的条目对象。大多数可视实例都称作*工作条目*，并且充当 AssemblyLine（AL）中的主数据载体。这是用于将数据沿着 AL 从一个组件传送到下一个组件的存储区。

工作条目通过预注册变量 *work* 可用于脚本编制，从而使您可以直接访问由 AssemblyLine 处理的属性（和其值）。此外，由工作条目携带的所有属性都显示在配置编辑器中，位于 AssemblyLine 的 AssemblyLine编辑器窗口中“属性映射”区域的**工作属性**标题之下。

条目类型

AssemblyLine 中有许多依据条目数据模型的数据对象。它们是:

Work 这是在 AssemblyLine 的组件之间移动并在其间传送数据的前述条目。其预注册的变量名称为 *work*，且可用于在几乎任何位置进行脚本编制[6]。

Conn 这是类似条目的对象，您在工作条目中提供数据或部分数据之前，连接器将此

6. 在未使用初始工作条目调用您所使用的 AssemblyLine 时，*work* 对象将不可用，直到 prolog 挂钩*之后*为止。在 Prolog 挂钩中，您可以拥有如下代码:

```
if (work != null) {
 // An Initial work Entry has been provided, we can get values from there
 .... some code
} else {
 // No initial work Entry has been provided
 ... some other code
}
```

对象用作所连接系统和 AssemblyLine 之间的媒介。在 Conn 和 Work 之间移动数据的过程称为“属性映射”。其预注册的变量名称为 *conn*，且其在连接器和函数组件中的许多挂钩内可用于脚本编制。

Current®

在 Update 方式的连接器中的某些挂钩内提供了这个类似条目的对象，在对来自已连接系统的数据应用任何更新之前，此对象将保留此数据。其预注册的变量名称为 *current*。

Error 在已发生某些错误条件且已调用相关错误挂钩时，此类似条目的对象仅存在于组件的某些挂钩中。其包含有关抛出的实际异常的信息，可能带有某些其他变量和数据，使您可以查明导致错误的准确原因。其预注册的变量名称为 *error*。

参考 中的“连接器流程”图将向您显示在何种情况下其中有哪些对象可用。

另见

第 40 页的『内部数据模型: 条目、属性和值』.

第 2 章 IBM Security Directory Integrator 中的脚本编制

IBM Security Directory Integrator 向其用户提供了高度灵活的引擎，您可以从配置编辑器的用户界面控件定制引擎，也可以通过定制逻辑的脚本编制来定制引擎。用户界面控件提供在较高级别控制数据流的方法，而脚本编制为您提供在任何级别控制数据流的几乎所有方面的能力（包含覆盖标准 IBM Security Directory Integrator 处理）。*system* 对象中提供了特殊函数，以在 AssemblyLine 条目上重新迭代、跳过连接器以及启动新的 AssemblyLine。用于实施该定制逻辑的脚本语言是 JavaScript。

IBM Security Directory Integrator 提供了一些现成可用的工具，可快速搭建起集成解决方案的框架。但是，对于所有最普通的迁移工作，您仍然需要通过编写 JavaScript 来定制和扩展该产品的内置行为。

IBM Security Directory Integrator 是纯 Java。只要在您向 IBM Security Directory Integrator 发布命令，使用组件和对象，或者处理流中的数据时，您都是在与 Java 对象打交道。IBM Security Directory Integrator 使用 IBM Java V7.0.4。

另一方面，您的定制却是在 JavaScript 中实现的，并且这两种表面相似实际根本不同的编程语言的紧密结合值得更深入地研究。

具有使用 JavaScript 的经验将会十分有帮助。提供的实例可以增加您的经验。但是，本手册未对 JavaScript 进行具体说明，而仅仅介绍了它在 IBM Security Directory Integrator 中的应用。您将需要保证您的 JavaScript 引用材料在其他地方的安全。

在网络上，有很多可以通过商业途径将引用指南用于 JavaScript 以及文档、教程和示例。请注意，尽管 Web 上有那么多 JavaScript 内容，但是它们都是相关于梅花和自动化 HTML 内容的方面。您只需要关心 JavaScript 核心语言本身，这方面内容在以下链接中描述：http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.5/guide/index.html

在这个站点上还有一个方便的链接，用于下载 HTML 格式的参考文档以便在本地安装。有关 JavaScript 的一本优秀手册是 *The Definitive JavaScript Guide*，第四版，作者 David Flanagan (O'Reilly)。

您还需要 Java 的 Javadocs，因为所有 IBM Security Directory Integrator 对象以及解决方案中的数据值都以 Java 对象的形式提供。这些文档在线提供，可通过以下 URL 访问：http://docs.oracle.com/javase/1.6.0/docs/api/index.html

J2SE 文档本身可通过以下地址访问：http://docs.oracle.com/javase/1.6.0/docs/index.html

当需要向 AssemblyLine 添加定制处理时，需要脚本编制。脚本编制可以有所帮助的示例包括以下任务:

属性操作或计算
: 您需要根据一个或多个输入属性计算输出属性的值。

数据过滤
: 想要仅处理与特定条件集匹配的条目。

数据一致性或有效性检查
: 需要报告或更正无效的数据值。

流量控制
想要覆盖您要使用的连接器的更新操作。

初始化 想要在 AssemblyLine 启动之前运行一些初始化过程。

以上提到的每种情况（还有很多其他未提到的情况）都需要脚本编制。

示例

请转至 IBM Security Directory Integrator 安装的 `examples/scripting` 子目录。

内部数据模型: 条目、属性和值

当 IBM Security Directory Integrator 组件访问已连接系统的信息时，它们使用 Java 对象将数据从系统特定类型转换为内部表示。在输出时，组件进行反向转换，将这个内部数据模型转换为目标系统的本机类型。当您希望将数据传入和传出 AssemblyLine 时，会用到这一相同的内部表示。因此，理解 IBM Security Directory Integrator 内部数据模型的工作方式非常重要。

让我们详细了解一下，当组件接收到数据值时，将使用正在读取的属性名称来创建对应的 IBM Security Directory Integrator *Attribute* 对象。数据值本身（将转换为相应的 Java 对象，例如，`Java.lang.String` 或 `java.sql.Timestamp`，并分配给 Attribute。如果您查阅 IBM Security Directory Integrator API 文档，您将会看到 Attribute 对象提供了大量的有用方法，例如 `getValue()`、`addValue()` 和 `size()`。这可使您直接从脚本创建、枚举和处理 Attribute 的值。您也可以根据需要实例化新的 Attribute 对象，如下例所示，该示例演示了对于目录的 objectClass 属性的高级映射:

```
var oc = system.newAttribute( "objectClass" );

oc.addValue( "top" );
oc.addValue( "person" );
oc.addValue( "organizationalPerson" );
oc.addValue( "inetOrgPerson" );

ret.value = oc;
```

属性本身收集在称为 *entry object* 的数据存储对象中。 *Entry* 是系统中主要的数据携带者对象，通过将重要的条目对象注册为脚本变量，IBM Security Directory Integrator 使您可访问这些对象。用于在 AL 组件之间（以及 AssemblyLine 之间）传递数据的一个主要示例是 AssemblyLine 中的 Work 条目对象。此条目对象对于每个 AssemblyLine 都是本地的，并且可作为脚本变量 *work*。

IBM Security Directory Integrator 提供了一些在使用 JavaScript 时使用的方便快捷的功能，因此上面特定的高级映射可以简单地编码为以下内容:

```
ret.value = [ "top", "person", "organizationalPerson", "inetOrgPerson" ];
```

高级映射功能支持 JavaScript 数组和条目，以传递多个属性值。

例如，在输入属性映射（该映射使得映射的属性在返回时显示在 *work* 条目中）中，假定您为“last”属性进行了如下分配:

```
ret.val = anentry;
```

。让我们进一步假定 *work* 开始时为空，而 *anentry* 则包含属性“cn”、“sn”和“mail”。

属性映射后，*work* 将包含“cn”、“sn”和“mail”属性，**不是**一个值为“anentry”的“last”属性。其实，在属性映射中，当属性映射返回 Entry 对象时，它将与正在接收的条目（*work* 或 *conn*）合并，具体取决于映射的类型（输入或输出）。

注: 在 IBM Security Directory Integrator 中，利用分层对象，通过先将条目封装在属性中，然后进行属性映射，可避免此行为。例如，

```
// this is the entry to return
e = system.newEntry();
e.setAttribute("some", "value");

// Create an Attribute object. We don't need to provide a name since the mapping
will use current map's name.
attr = system.newAttribute(null);

// add the entry to the Attribute object and return that instead of the Entry object
attr.addValue(e);

return attr;
```

如果这是作为“last”属性的高级属性映射输入的，那么进行属性映射后，*work* 条目现在将包含“last”属性。此属性为条目，依次由两个属性“some”和“value”组成。

。

参阅 Javadocs，您将会看到条目对象为处理条目及其属性和值提供了各种函数，包括 `getAttributeNames()`、`getAttribute()` 和 `setAttribute()`。如果您希望创建属性，并将其添加到 AssemblyLine 工作条目中，那么您可以使用以下脚本，例如，在挂钩或脚本组件中:

```
var oc = system.newAttribute( "objectClass" );

oc.addValue( "top" );
oc.addValue( "organizationalUnit" )

work.setAttribute( oc );
```

请注意，对于这种情况，我们**不**具有使用 JavaScript 数组来设置值的选项，具体如下:

```
oc.addValue( ["top", "organizationalUnit"] );  // Does not work like Advanced Mapping
```

此段代码将使 `oc` 属性得到一个值，也就是一个字符串数组。

Entry 对象还可以包含*属性*。Property 和 Attribute 一样也是数据容器，但是 Property 仅仅是单值。 “属性”用来存储数据内容，而“特性”保存参数信息，这样可使您将这些信息分开。Property 不会出现在属性映射选择或工作条目列表中，但是很像 Attribute，可以从 script.entry 函数访问，如 `getProperty()` 和 `setProperty()` 可用于此目的，这些函数直接处理 Property 值，就像 Attribute 值一样，Property 值可以是任何类型的 Java 对象。与处理 Attribute 所不同的是，不存在 Property 中间对象。

在很多情况下，您可以将数据模型限制为不包含属性或包含一个及以上属性的条目，每个属性不包含值或包含一个或更多值 – 这就是平面模式。

这是 IBM Security Directory Integrator 的一个优势: 简化和协调数据表示和模式。当您需要处理结构更为复杂的信息时，它也提出了一个挑战。但是，由于属性值可以是任何类型的 Java 对象，包括其他条目对象（具有其自己的属性和值），因此 IBM Security Directory Integrator 可允许您处理具有层次结构的数据。

这种用于处理分层对象的更详尽和结构化的方法在 第 42 页的『使用分层 Entry 对象』中进行了描述。

使用分层 Entry 对象

使用分层结构化数据的备选方法是利用对 IBM Security Directory Integrator Entry 对象中分层对象的支持。

与早期版本不同的是，IBM Security Directory Integrator V7.1.1 和后续版本 支持分层 Entry 对象的概念。Entry 对象表示层次结构的根，每个属性都表示该层次结构中的节点。遵循此逻辑，每个属性的值是层次结构中的叶子。还存在用于遍历层次结构的 API。此 API 是 DOM 3 规范的不完全实现。仅实现了该规范中的少数类:

- `Org.w3c.dom.Document` - 通过 Entry 类实现。
- `Org.w3c.Element` - 通过 Attribute 类实现。
- `Org.w3c.Attr` - 通过 Property 类实现。
- `Org.w3c.Text` 和 `org.w3c.CDATASection` - 通过 AttributeValue 类实现。

这些类是 DOM 规范提供的类的最小集合，用于表示分层数据。由于向后兼容性的原因，V7.0 之前的 API 不能识别层次结构（例如，它无法访问/修改/除去子元素）。这是只有 DOM API 才能处理分层结构的原因。

要保持条目结构可以向后兼容，缺省情况下，条目始终使用平面属性。条目仅在您调用新近提供的一个 DOM API 后才会根据需要变为分层形式。这样，只允许了解条目的分层特性的组件使用此功能；而其余组件为了保持正常运行，不需要进行更改。从 IBM Security Directory Integrator v7.0 开始，引入了一种新的名称表示法，以便用户可以更轻松地创建分层树。其中包含点的每个名称都视为由简单名称组成的组合名称；这些名称通过点分隔。将这样的组合名称传给分层条目时，会将该名称分解为简单名称，并构建由该组合名称描述的层次结构。

例如，如果运行以下 JavaScript 代码:

```
// create a new empty Entry object
var entry = new com.ibm.di.entry.Entry(true);
// create a new branch of 2 levels
entry.setAttribute("firstLevelChild.secondLevelChild", "level2Value");
// finds the already existing branch and creates a new node on level 3
entry.setAttribute("firstLevelChild.secondLevelChild.thirdLevelChild", "level3Value");
```

将创建以下结构:

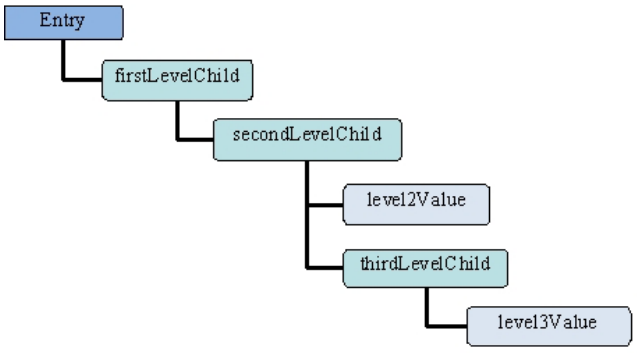


图 1. 简单分层条目

注: 仅当条目结构转换为分层结构时，这些名称会分解为简单名称，了解这一点非常重要。例如，如果条目为平面结构，并且仅使用仍然会创建相同旧平面结构的旧方

法:

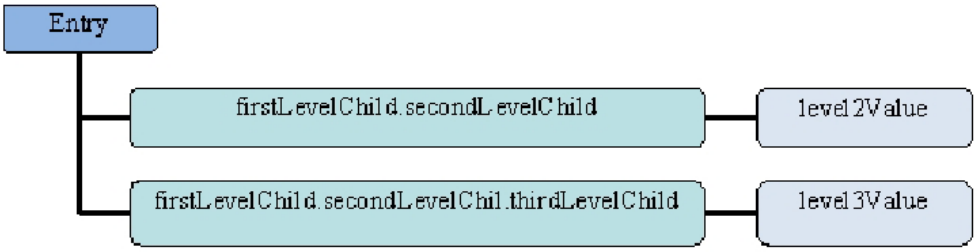


图 2. 传统的平面条目

在有些情况下可能需要在条目属性的名称中使用点，这样 Entry 对象也理解转义字符（当前仅支持 \\ 和 \.）。

注: 在脚本中使用时，反斜杠应该用另一个反斜杠正确转义！
例如，如果需要以下层次结构:

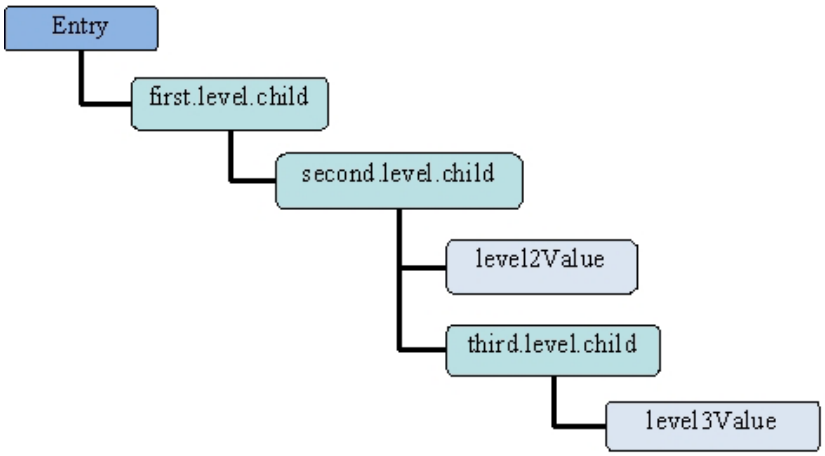


图 3. 另一个简单的分层条目

以下脚本将创建该条目:

```
var entry = new com.ibm.di.entry.Entry(true);
entry.setAttribute("first\\.level\\.child.second\\.level\\.child", "level2Value");
entry.setAttribute("first\\.level\\.child.second\\.level\\.child.third\\.level\\.child", "level3Value");
```

为了在隐式使用分层条目时保持与 IBM Security Directory Integrator 以前发行版的向后兼容性，Attribute 类和 Entry 类中公开的所有旧方法都进行了稍微的更改。例如，`Entry.getAttributeNames()` 方法将返回树中可用叶子的完整路径的数组。关于以上结构，getAttributeNames 方法返回以下数组:

```
["first\.level\.child.second\.level\.child", "first\.level\.child.second\.level\.child.third\.level\.child"]
```

`Entry.size()` 方法返回 `getAttributeNames()` 方法所返回的数组中的元素总数。在这里，size() 方法将返回 2（整棵树中的叶子数量）而不是 1（鉴于 Entry 对象只有一个属性作为子对象，您可能会如此预期）。

这是因为 getAttributeNames() 和 size() 方法都只处理平面结构。为了得到子元素的实际大小，您将需要按如下所示使用 DOM API: `Entry.getChildNodes().getLength();`

如果条目是平面结构，那么旧 API 的行为将与先前发行版的行为相同。

Attribute 对象

Attribute 对象具有创建分层结构的增强功能。这些结构遵循 DOM 规范，这是 Attribute 对象了解 XML 名称空间概念的原因。

为了为分层功能提供新方法，还必须扩展 Attribute 类。getValue/setValue/addValue 方法也是向后兼容的，将仅返回特定元素具有的值。此处的差别是通过 DOM API 访问每个值时，会将每个值封装在 AttributeValue 类中，并将其视为 Text 节点。为了得到 Attribute 的子元素（例如，Attribute 和 AttributeValue），必须使用 DOM API。

访问 Entry 的任何 DOM 方法时，Entry 的子元素 Attribute 也能够将它的结构切换为分层结构。与 Entry 类不同，Attribute 类隐式执行此操作，不会提供显式执行此切换的方法。

IBM Security Directory Integrator v7.0 支持对服务器脚本编制功能的扩展，从而可轻松访问复杂的结构。请参阅第 45 页的『脚本中的导航』部分了解更多详细信息。

AttributeValue 对象

此类表示分层树中的值。按照 DOM 规范，树中的值始终为 String。但是，在前几个版本中，AttributeValue 类包含任何种类的对象。这在当前版本中也是有效的。AttributeValue 对象的唯一差别在于通过 DOM 访问时，它将返回所包含对象的字符串表示形式。为了使 AttributeValue 类能够表示节点，同时具有 DOM 规范定义的值，那么它必须实现 org.w3c.dom.Text 或 org.w3c.dom.CDATASection 接口。AttributeValue 实现这两个接口，并可以根据您的需要表示其中任何一个节点。

Property 对象

Attribute 可以包含零个、一个或多个 Property 对象。Property 类实现 org.w3c.dom.Attr 接口，因此根据 DOM 概念表示属性。使用这些属性，您可以声明与 XML 概念相关的前缀/名称空间。

传送对象

将 Attribute 从一个条目映射到另一个条目会始终复制源 Attribute。

例如:

```
entry.appendChild(conn.getFirstChild());
// or
entry.setAttribute("name", conn.getFirstChild());
```

即使当 Attribute 不是 Entry 的第一层子元素时，也仍然进行复制。这也可以通过脚本实现:

```
entry.a.b.c.d.appendChild(conn.e.f.g);
```

为了使 Attribute 对象在条目之间移动而不进行克隆，您将需要先使其与原父元素断开连接，然后将其连接到新的父元素。

例如:

```
var src = entry1.b.source;
entry1.b.removeChild(src);
entry2.a.target.appendChild(src);
```

将 Attribute 对象从一个父元素移动到相同 Entry 中的另一个父元素时，将自动移动 Attribute。且不执行克隆。

例如:

```
entry.a.target.appendChild(entry.b.source);
```

在此示例中，“source”属性与其父属性（“entry.b”）断开连接，然后连接到了“entry.a.target”属性。且不执行克隆。

如果您不希望从源除去 Attribute 对象，那么您可以通过以下方式附加 Attribute 的副本:

```
entry.a.target.appendChild(entry.b.source.clone());
```

脚本中的导航

IBM Security Directory Integrator ScriptEngine 使您能够仅通过名称引用条目属性就可访问条目的属性；例如，`entry.attrName` 返回名称为 *attrName* 的属性。

1. JavaScript 引擎基于请求名称所依据的上下文对象来解析名称。例如，如果执行 `entry.a` 的调用，那么 *entry* 名称将是上下文对象，*a* 是要解析的子对象的名称。JavaScript 引擎使用从左至右的解释方式评估各上下文对象，直至解析了最后一个对象为止。根据下图，使用此过程解析以下 `entry.a.b.c` 调用: 查找要将其用作第一步的上下文对象的 *entry* 对象。
2. 搜索名称为 *a* 的上下文对象。*entry* 对象仅具有一个名称为 *a* 的子元素。该子元素视为下一步的下一个上下文对象。
3. 搜索名称为 *b* 的上下文对象。该上下文对象具有名称为 *b* 的两个子元素。将它们放在列表中，返回该列表。
4. 最后的操作是在以前的操作中返回的列表中搜索名称 *c*。列表中的每个元素至少具有一个这样的子元素。获取所有这些元素，将它们放在列表中，这是解析完整表达式的实际结果。

以下 Entry 对象示例图说明了此操作:

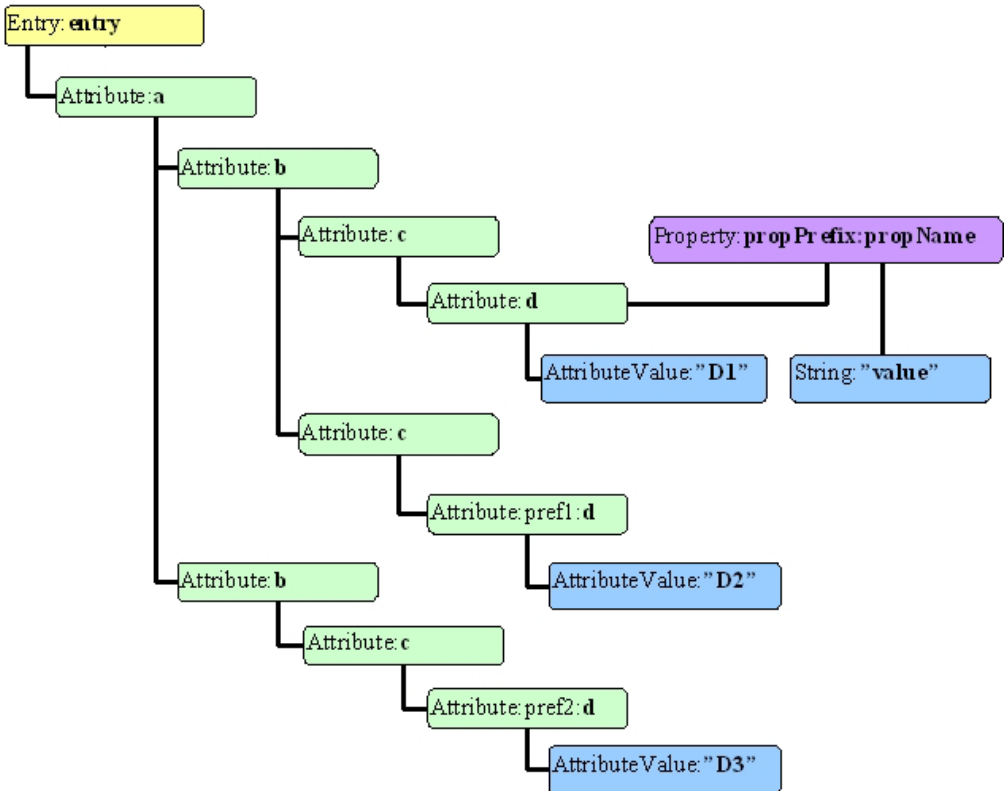


图 4. 分层 Entry 对象示例

IBM Security Directory Integrator 提供的脚本引擎允许在子元素解析过程中使用更多任意名称。例如，如果子代的名称包含点，那么可以使用方括号语法引用该名称，如下所示:

```
work["{namespace}name:Containing\.invalid\.charaters"]
```

请注意，点会进行转义，以表示其属于局部名的一部分，脚本引擎不得将其作为路径分隔符进行处理。

最终结果可能不同，具体取决于对其执行操作的当前上下文对象。通过对下列对象实施若干项增强功能，IBM Security Directory Integrator 遵循 JavaScript 引擎提供的标准对象操控:

条目 当上下文对象为此类型的实例时，名称解析机制将查找以下语法:

- `@<name>` - 搜索 Entry 对象中具有指定名称的属性。解析得到的对象可能为 NULL，或者为映射给该属性的对象。
- `<prefix>:<localName>` - 搜索 Entry 对象中前缀等于 <prefix> 且局部名等于 <localName> 的子元素。解析得到的对象可能为 NULL，或者为现有的 Attribute 对象。
- `<localName>` - 搜索 Entry 对象中局部名等于 <localName> 的第一个子元素。解析得到的对象可能为 NULL，或者为 Attribute 对象。
- `{namespaceURI}<localName>` - 搜索 Entry 中属于指定 namespaceURI 且局部名与指定名称相同的第一个子属性。

注: 如果提供了前缀，那么将忽略该前缀，名称解析机制将仅查找指定的 namespaceURI 和 localName。解析得到的对象可能为 NULL，或者为 Attribute 对象。

属性 当上下文对象为此类型的实例时，名称解析机制将查找以下语法:

- `@<prefix>:<localName>` 和 `@<localName>` - 搜索 Attribute 中具有相同前缀和/或局部名或者仅具有指定局部名的 Property 对象。如果没有属性与指定名称匹配，那么返回 NULL，否则返回单个 Property 对象。
- `@{namespaceURI}<localName>` - 搜索 Attribute 中属于指定 namespaceURI 且局部名与指定名称相同的 Property。

  注: 如果提供了前缀，那么将忽略该前缀，名称解析机制将仅查找指定的 namespaceURI 和 localName。解析得到的对象可能为 NULL，或者为 Property 对象。
- `[<index>]` - 指定 Attribute 中要检索的值的位置。使用此表示法，您无法访问此 Attribute 的子元素。返回 NULL，或者返回指定位置的对象。
- `<prefix>:<localName>` 和 `<localName>` - 搜索 Attribute 中具有指定前缀和/或局部名的子元素。返回 NULL，或者返回具有指定名称（如果只有一个名称）的子 Attribute，或者返回包含与该条件匹配的所有子 Attribute 的 NodeList。
- `{namespaceURI}<localName>` - 搜索 Attribute 中属于指定 namespaceURI 且局部名与指定名称相同的所有子 Attribute。

  注: 如果提供了前缀，那么将忽略该前缀，名称解析机制将仅查找指定的 namespaceURI 和 localName。解析得到的对象可能为 NULL、单个 Attribute 对象或者为包含与搜索名称匹配的所有 Attribute 的 NodeList。

NodeList

当上下文对象为此类型的实例时，名称解析机制将查找以下语法:

- `[<index>]` - 指定 NodeList 中要检索的元素的位置。返回 NULL，或者返回指定位置的对象。如果索引超出了范围，那么会抛出异常。
- `@<prefix>:<localName>` 和 `@<localName>` - 在 NodeList 的每个元素中搜索具有相同前缀和/或局部名的属性。返回 NULL、Property 对象（如果只找到一个对象）或在 NodeList 中找到的所有 Property 对象的列表。
- `@{namespaceURI}<localName>` - 搜索每个 Attribute 中属于指定 namespaceURI，且局部名与指定名称相同的 Property。

  注: 如果提供了前缀，那么将忽略该前缀，名称解析机制将仅查找指定的 namespaceURI 和 localName。解析得到的对象可能为 NULL，或者为 Property 对象（如果只找到一个对象），或者为在该 NodeList 中找到的所有 Property 对象的 NodeList。
- `<prefix>:<localName>` 和 `<localName>` - 搜索 NodeList 的每个元素中具有相同前缀和/或局部名的子元素。返回 NULL、Attribute 对象（如果只找到一个对象）或在 NodeList 中找到的所有 Attribute 对象的 NodeList。
- `{namespaceURI}<localName>` - 搜索每个 Attribute 中属于指定 namespaceURI 且局部名与指定名称相同的所有子属性。

注: 如果提供了前缀，那么将忽略该前缀，名称解析机制将仅查找指定的 namespaceURI 和 localName。解析得到的对象可能为 NULL、单个 Attribute 对象或者为包含与搜索名称匹配的所有 Attribute 的 NodeList。

您现在具有以下选项:

1. 能够访问所有 *d* 元素，方法是从顶部开始引用这些元素；例如，`entry.a.b.c.d` – 这会返回具有与该路径匹配的所有 *d* 属性且类型为 NodeList 的对象。在我们的示例中，这将返回所有三个 *d* 元素。
2. 能够访问属性，方法是指定前缀和局部名；例如，`entry["a.b.c.pref1:d"]` – 这将仅返回单个 Attribute，其前缀为“pref1”。
3. 能够使用 [ ] 表示法，访问 NodeList 的每个 Attribute，例如，`entry.a.b.c.d[0]` – 这会返回单个 Attribute，即以上结构中的第一个 *d* 元素。
4. 能够使用 [ ] 表示法浏览这些元素；例如，`entry.a.b[0].c.d` – 这会返回 Attribute 的列表，但是此时它包含构成第一个 *b* 分支的所有 *d* 属性。
5. 能够使用 @ 表示法获取属性（即，使用 DOM 命名约定的 Element 的 Attribute）的特性；例如，`entry.a.b.c.d[0]["@propPrefix:propName"]` – 这会为我们返回一个包含该特性（即，根据 DOM 得到的属性值）的值的 String 对象。请注意，这里 propPrefix 和 propName 是使用冒号（“:”）分开的，如果属性有前缀，那么要找到该属性，必须指定前缀和名称。或者，可以使用名称空间和 propertyName 来查找具有前缀的特性。
6. 能够在使用用户希望执行的方法的名称搜索子元素之前调用 Attribute 的方法；例如，`entry.a.b.[0].getChildNodes()` – 这会返回一个 NodeList 对象，其中包含第一个 *b* 属性包含的所有 Attribute 值。
7. 能够通过名称访问条目属性；例如，`entry.attrName` – 这会返回映射到 *attrName* 键的 Attribute。
8. 能够使用 .@ 表示法访问条目属性；例如，`entry.@propName` – 这会返回作为属性映射到 *propName* 键的对象。
9. 能够通过指定子节点所属的名称空间来访问那些子节点。例如，`work.a.b[{someNamespace}c]` – 这会返回属于“someNamespace”名称空间的所有 *c* 对象。

注:

1. 如果属性名称与 Entry/Attribute 方法的名称相同，那么将调用该方法。如果您希望访问属性，那么必须像以前一样使用对象的方法（即，`Entry.getAttribute ("getAttribute");`。）
2. 如果使用平面条目，那么脚本引擎不会将条目转换为分层条目，除非指定了要查找的属性名称空间。例如: `entry["{ns}element"]`。脚本引擎会自动将条目转换为分层条目（如果上下文对象为 Attribute 类型）。例如在这种情况下: `entry.attr.child`。
3. 如果要查找的属性名称中包含点，并且条目是平面的，那么您必须使用括号表示法而不是点表示法，或者在解析子节点之前强制条目变为分层条目。例如，如果条目是平面的，那么您无法使用调用 `entry.http.body` 来访问属性 `http.body`。要使此情况发生，您将必须使用代码 `entry["http.body"]` 或调用 `entry.enableDOM()`，然后才能调用 `entry.http.body`。

4. 如果您希望多次使用从表达式检索的引用（例如 a.b.c.d），那么好的做法是将该引用分配给局部变量，因为每次对相同表达式重复求值将会导致检索相同引用的额外开销。
5. 例如，如果使用表达式 entry.a.b.c[0].d，那么会引用 Attribute 对象；但是如果使用 entry.a.b[0].c.d，那么会引用 NodeList 对象。为了识别所引用的对象，您可以检查对象的名称，例如:

```
var obj = a.b.c.d;
if (obj.getClass().getSimpleName().equals("Attribute") ) {
  // handle this as Attribute
} else {
  // handle this as a list of Attributes
}
```

例如，如果您需要处理表达式返回的每个元素（即使只返回一个元素），那么您可以像下面这样使用 for/in 循环结构:

```
for (obj in entry.a.b.c.d) {
  // the obj will be an object of type Attribute
}
```

现在可对 Entry 对象使用同样的用法，例如

```
for (obj in work) {
  // the obj will be an Entry's attribute
}
```

6. ScriptEngineOptions 也允许赋值。例如，以下表达式是有效的:

entry.a.b.c.d[0][″@propPrefix:propName″] = ″new value″;
这会更改属性的值。如果属性不存在，那么将会创建该属性。

entry.a.b.c.d[0][0] = ″new value″;
这会替换位置 0 处的属性值。

entry.a.b.c.d[0][1] = ″another value″;
这会向属性添加另一个值（如果索引与值数组的大小相同。）

entry.a.b.c.d[0][a.b.c.d[0].size()] = ″appended value″;
如果需要向对象列表中追加新值，但是不知道最后一个元素的位置，那么可以使用 Attribute#size() 以获取此元素具有的子代数。

new com.ibm.di.entry.Entry().@propName = ″someValue″;
这会在 Entry 对象中创建名称为 propName 的新属性（如果该属性不存在），并将其值设置为字符串“someValue”。

entry.a.b.c[1] = ″value″;
这会将 *entry.a.b.c* 解析为 NodeList，并自动将新的 String 对象作为值添加到已解析 NodeList 对象的第二个元素中。例如，如果 *entry.a.b.c* 解析为属性，那么新的 String 值将替换已解析的 *c* 属性的第二个值。

entry.a.b.c[2][″pref3:d″] = ″value″;
这会将新的 *pref3:d* 子元素添加到 *entry.a.b.c* 列表的第三个 *c* 属性中。请注意，*entry.a.b.c[2]* 解析为 Attribute 和列表。

entry.a.b.c[″{namespaceURI}:d″] = ″myTextValue″;
这会将值设置为 *c* 的第一个子属性，它的局部名等于 *d*，并且属于名称空间 namespaceURI。如果找不到这样的属性，那么会创建一个新的属性并为其

赋值。创建属性时，您也可以将名称空间与前缀关联。在本例中，可以使用以下代码完成此操作：`entry.a.b.c["{namespaceURI}prefix:d"] = "myTextValue";`。

entry[″{http://ibm.com/xmlns/}first.second.third″] = null;

这将首先尝试从名称空间“http://ibm.com/xmlns/”中查找局部名为“first”的元素。如果失败，那么将创建该元素，然后将尝试解析其子元素“second”。如果失败，那么将创建该子元素并将其名称空间设置为第一个元素的名称空间。最后，将尝试对第三个元素进行解析。如果失败，那么将创建最后一个元素，但其没有任何值。这与 `entry["{http://ibm.com/xmlns/}first.{http://ibm.com/xmlns/}second.{http://ibm.com/xmlns/}third"] = null;` 等效

使用脚本创建上述结构

```
// create a new entry that will hold the structure
var entry = new com.ibm.di.entry.Entry();
// create the first branch
var d = entry.newAttribute("a.b.c.d");
// create a new property and assign it a value
d["@propPrefix:PropName"] = "value";
// create a new value of the d Attribute
d[0] = "D1";
// append a new value for the entry.a.b attribute
entry.a.b.appendChild(entry.createElement("c"));
// create a new Attribute d with a prefix on the second c attribute,
// and assign the string "D2" as a first value
entry.a.b.c[1]["pref1:d"] = "D2";
// create a new child of the a Attribute
entry.a.appendChild(entry.createElement("b"));
// choose the second attribute from the entry.a.b NodeList and create a new child named c
entry.a.b[1].appendChild(entry.createElement("c"));
// create the d child of the c Attribute
entry.a.b[1].c["pref2:d"] = "D3";
```

使用 XPath 导航

Entry 类提供了根据 XPath 表达式导航和检索数据的便利方法。使用 XPath 查询 Entry 中的数据比使用脚本中的任何简单导航都高级得多。为了达到相同效果，在单个表达式中实现搜索和/或值匹配逻辑比编写多行脚本简单得多。

Entry 类提供了以下方法:

- `NodeList getNodeList (String xPath);`
- `Attribute getFirstAttribute(String xPath);`
- `String getStringValue(String xPath);`
- `Number getNumberValue(String xPath);`
- `Boolean getBooleanValue(String xPath);`

使分层结构变平

使用分层数据时，有时候需要使层次结构的节点变平。为此，您可以编写用于遍历树的脚本，或使用以下方法之一:

- `getElementsByTagName(String namespace, String localName);`
- `getElementsByTagName(String tagName);`

这些方法将遍历树并搜索具有指定名称和名称空间的元素。DOM API 允许这些方法在名称和名称空间中接受字符“*”。该字符表示通配符，用于与任何元素名称/名称空间匹

配。在名称中使用该字符可通过返回 NodeList 中的所有 Attribute 节点，从而使树变平。应该注意，分层结构并未更改。返回的 NodeList 只是在树中找到的 Element 节点的容器。

如果您对第 45 页的『脚本中的导航』部分的结构中的条目运行以下脚本:

```
var list = entry.getElementsByTagName("*");
```

那么，*list* 变量将包含以下结构:

```
列表
   |
  + a
   |
  + b
   |
  + c
   |
  + d
   |
  + c
   |
  + pref:d
   |
  + b
   |
  + c
   |
  + pref2:d
```

异常

在以下情况下会抛出异常:

- 如果调用了方法 `entry.appendChild(newAttr)`，并且条目已包含具有 newAttr 对象（已传递给该方法）的名称的 Attribute。
- 如果对于由 Document/Node/Element 定义且由 Entry/Attribute 类实现的任何方法传递了意外参数。
- 如果提供的脚本引用了 Attribute/NodeList 中不存在的索引，那么会抛出 ArrayIndexOutOfBoundsException。

将脚本编制集成到解决方案中

如已经说明的，无论何时需要在集成解决方案中定制处理，都可使用脚本。使用 IBM Security Directory Integrator 的最好方法是将该定制处理分成两个类别: 属性变换和流量控制。

注: 这只是约定，并非系统强制执行的限制或规则。在数据流中的某个可标识点（例如，任何处理开始前、处理特定条目前、发生故障后等）不可避免地需要定制数据处理，因此通过将代码放置在尽可能靠近此点的地方，可以使解决方案更易于理解和维护。

处理属性转换的合理位置是在**属性映射**中，“输入”和“输出”都可。如果需要计算脚本编制逻辑或 AssemblyLine 下游中其他连接器必需的新属性，最佳实践是在**输入映射**中完成此操作（如果可能）。或者，如果您必须为了单个输出源而变换属性，那么可以通过将这些属性放置到相关连接器的**输出映射**中，来避免混淆带有特定输出变换的**工作**条目对象。

其他类别的定制逻辑和流控制最好在自动工作流中需要该逻辑的点处调用的挂钩中实现。可以从配置编辑器很容易地访问这些控制点。实现定制处理只需要识别正确的控制点并在相应的编辑窗口中添加脚本。

AssemblyLine **脚本组件**（脚本编写代码的独立块）还为您提供了创建自己的定制处理的位置，然后使您能够将代码重新放置在 AssemblyLine 中。虽然在测试和调试的过程中频繁地使用“脚本组件”，它们还是可以在生产配置中扮演重要角色。只需记住要清楚地命名您的组件，并在脚本自身中包含某个文档以[7]说明为何在“脚本组件”中而不是在**属性映射**或**挂钩**中实施此逻辑。

正确地确定相应控制点（输入脚本的位置）和将脚本的作用域限制为只涵盖与控制点相关联的单个目标，这两者同样都很重要。如果您使逻辑单元保持相互独立，那么可将其复用的几率更大，而当组件在其他上下文中重新排序或复用时，其可能崩溃的几率更小。构建可再用的代码的一种方法是通过在**脚本库**（或 **Prolog** 挂钩）中创建自己的函数来实现频繁使用的逻辑，而不是复制和粘贴相同代码至多个位置。

这里总结了一些最佳实践，请您在构建解决方案时谨记:

- 在“属性映射”中执行属性操作。
- 将流量控制（过滤、验证、分支等）放置在挂钩中，并将 AssemblyLine 脚本组件放置在必要的位置。
- 尽可能多地使用自动行为；例如，AssemblyLine 工作流和连接器方式。
- 通过使 AssemblyLine 保持简短和集中来简化您的解决方案。
- 将常用逻辑置于离散的块中；例如，“资源”部分中的脚本。
- 考虑重用。

值得再次提及的是，虽然先前描述的方法都是最佳实践，但您可能会遇到必须偏离已建立的约定的情况。如果是这种情况，随着时间的过去，您的解决方案文档对于维护和提高您的工作是至关重要的。

使用脚本编制控制执行

引擎展示了大量类和对象，可以从 AssemblyLine 中用户创建的脚本访问、读取和修改这些类和对象。这些对象表示任何时候 AssemblyLine 的状态和整个 IBM Security Directory Integrator 环境的状态。通过修改任意这些对象，可以修改 IBM Security Directory Integrator 环境并因此影响继承过程的执行。

注: 更改可应用于组件或 AssemblyLine 的实例。还可以对可运行参数（例如系统或 Java 参数）进行更改。还可以对配置文件或配置进行更改。在这种情况下，配置对象的新实例将反映这些更改。

有关全局对象的更多信息，请参阅 IBM Security Directory Integrator 产品附带的 Javadoc（通过选择配置编辑器中的**帮助 > Javadoc**）。

可用的所有类和实例的描述可以在安装软件包中找到。

通过了解所展示的类和接口，可以更好的了解 IBM Security Directory Integrator 引擎的元素以及元素之间的关系。

7. 在稍后某个时候必须再次访问您的代码时向他人以及您自己!

使用变量

将被处理的数据的标准容器对象（条目对象）与 JavaScript 提供给您的其他通用变量和数据类型区分开来是很重要的，您自己创建时也要进行这种区分。就 IBM Security Directory Integrator 解决方案内的脚本中可以放置的内容而言，您的创造性和脚本语言的功能是仅有的约束。但是，当在数据流上下文中操作数据时，必须注意并使用条目对象的结构。

条目对象带有属性，这些属性本身就是数据值的容器。属性值本身就是对象（`java.lang.String`、`java.util.Date` 和更复杂的结构）。属性值甚至可以是另一个带有其自己的属性和值集合的条目对象。IBM Security Directory Integrator 的工作是了解数据如何存储在已连接系统中，以及了解这些本机类型如何转换到系统的数据表示中（处于 Java 对象中）或从其中进行转换。

如果知道属性值的类，就可以成功访问并解释这个值。例如，如果 `java.lang.String` 属性包含想要用作浮点的浮点值，那么必须首先将该值（通过脚本编制语言的方法）手动转换为某种数字数据类型。

在创建变量或过程，而它们与集成过程中的数据流和可用的全局对象不直接相关时，应用以下原则：您可以声明并使用脚本语言支持的任何变量（对象）。这些变量的用途是帮助您实现与控制点（您在其中编制脚本）相关联的特定目的。变量必须只作为临时缓冲区，且不试图影响 IBM Security Directory Integrator 环境的状态。

使用属性

在 AssemblyLine 生命周期内，IBM Security Directory Integrator Server 提供了许多与 AssemblyLine 执行环境相关的组件属性，您可以通过脚本组件或组件挂钩在脚本中查询这些属性。甚至可以设置一个属性（*lastCallStatus*）。

通过使用 AssemblyLine 对象（例如连接器）访问属性并调用其方法 `get(property_name)` 来抽取 property_name 值；此外，使用 `put(property_name, property_value)` 方法将属性设置为期望值。设置属性时，属性名称或属性值不能为空；如果有一个为空，将抛出具有相应消息的异常。

可用的属性集合如下所示:

表 6. AssemblyLine 执行期间提供的组件属性

属性	用途
numErrors	发生的错误数。
numAdd	AssemblyLine 已添加的条目总数（由连接器在 AddOnly 方式下执行）。
numModify	AssemblyLine 已修改的条目总数（由连接器在 Update 方式下执行）。
numDelete	AssemblyLine 删除的条目总数（由连接器以 Delete 方式执行）。
numGet	AssemblyLine 已检索的条目总数（由连接器在 Iterator 方式下执行）。
numGetTries	AssemblyLine 尝试检索条目的总次数（由连接器以 Iterator 方式执行）。
numGetClient	接受的客户机总数（适用于 Server 方式的连接器）。
numGetClientTries	AssemblyLine 已尝试获取下一个已连接客户机的总次数（由连接器在 Server 方式下执行）。
numCallreply	AssemblyLine 已执行的调用/回复总数（由连接器在 CallReply 方式下执行）。

表 6. AssemblyLine 执行期间提供的组件属性 (续)

属性	用途
numLookup	AssemblyLine 已执行的查找操作总数（由连接器在 Update/Delete/Lookup 方式下执行）。
numNoChange	AssemblyLine 已处理但未更改的条目总数。
numSkipped	AssemblyLine 跳过的条目总数。
numIgnored	AssemblyLine 已忽略的条目总数（由连接器在 Update/Delta 方式下执行）。
lastCallStatus	包含 AL 执行的状态。它不仅是只读属性，该属性可以进行修改。该属性的值是“fail”或“success”，这取决于 AL 执行。
lastConn	来自上个连接器操作的 Conn 条目。首个连接器操作之前，lastConn 具有空值。
lastError	作为 Java 对象的上一个错误。
hooksInvoked	上次调用组件时调用的挂钩名称的 java.util.List。这些名称是内部名称。
success	如果上次操作成功，该属性将设置为 TRUE，否则将设置为 FALSE。
endOfData	如果迭代器组件达到了数据结尾，该值将为 TRUE，否则为 FALSE。更改该属性无效。

注: 如果尝试更改只读属性，那么将抛出具有相应消息的异常。

示例

要说明这些组件属性的用法，让我们假设您具有称为 *FS* 的文件系统连接器以及某些脚本组件。以下 JavaScript 代码位于 FS 的“GetNext Successful”挂钩中:

```
if(work.getString("ID") == null)
throw new java.lang.Exception("Missing ID");
//for the AL Cycle to execute properly I need an ID, so throw an Exception
```

而且该 JavaScript 代码位于 FS 的“DefaultOnError”挂钩中:

```
if(FS.get("lastError").getMessage().equals("Missing ID")) {
//I could fix this by adding an ID which would help AL execution
 work.setAttribute("ID", "SomeID"); //add the ID
 if(FS.get("fixErrors") == null) {
  var vector = new java.util.Vector();
  vector.add(FS.get("lastError"));
  FS.put("fixErrors", vector);//save all fixed errors in my custom property
 } else {//I have previously fixed similar error
  var vector = FS.get("fixErrors");
  vector.add(FS.get("lastError"));
  FS.put("fixErrors", vector); //save all fixed errors in my custom property
 }
 FS.put("lastCallStatus", "success");
} else {//I could not fix this error
 if(FS.get("notFixErrors") == null) {
  var vector = new java.util.Vector();
  vector.add(FS.get("lastError"));
  FS.put("notFixErrors", vector); //save all not fixed errors in my custom property
 } else {
  var vector = FS.get("notFixErrors");
  vector.add(FS.get("lastError"));
  FS.put("notFixErrors", vector); //save all not fixed errors in my custom property
 }
 FS.put("lastCallStatus", "fail");
}
```

最后，在脚本组件中，考虑以下代码:

```
main.logmsg("AL Cycle status: " + FS.get("lastCallStatus"));
//print the AL status for this AL Cycle
//I can also report all errors which have occurred
// during the AL Execution through my custom property, "vector"
```

脚本编制的控制点

AssemblyLine 中的脚本编制

AssemblyLine 提供了标准的预编程行为。如果想要使自己的行为与该标准行为不同，那么您可以通过用脚本编制的方法实施自己的业务逻辑来实现该目标。

脚本组件

除了连接器之外，你还可以将脚本组件添加到 AssemblyLine，方法是单击“AssemblyLine 编辑器”的“AssemblyLine 组件”窗口中相应组件或部分上的**插入组件 ... > 脚本 > 脚本**。将为 AssemblyLine 处理的每个条目启动一次脚本组件，并且可以将脚本组件放置在 AssemblyLine 中的任何地方。

注: 即使您在 AssemblyLine 中将“脚本组件”放置在迭代器之前，仍然会首先处理迭代器。

有关更多信息，请参阅第 19 页的『脚本组件』。

AssemblyLine 挂钩

AssemblyLine 的**挂钩**选项中将发现 AssemblyLine 挂钩（即，整体应用于 AssemblyLine 而不是任何单个组件的挂钩）。这些挂钩在每个 AssemblyLine 运行时，或在**关闭请求**情况（一旦告诉 AssemblyLine 其将被某些外部进程关闭）下都只执行一次。但是，如果多次启动 AssemblyLine （例如，通过使用 AssemblyLine 连接器），那么也将多次启动挂钩。

仅当定义挂钩的连接器运行时，才会评估和执行连接器中的挂钩（已定义且非空）。关于更多信息，请参阅第 60 页的『在连接器中编制脚本』。

服务器挂钩

服务器挂钩使您能够写入 JavaScript 代码以响应在服务器级别发生的事件和错误。与 AssemblyLine 和组件挂钩不一样的是，服务器挂钩存储在单独的脚本文件中。这些文件保存在当前解决方案目录中的 *serverhooks* 文件夹中，并且必须包含特别指定的脚本函数。IBM Security Directory Integrator 服务器和配置实例为了 IBM Security Directory Integrator 组件提供了一个可调用定制服务器级别挂钩的方法。服务器挂钩是脚本文件中定义的函数名。只需将脚本文件放入解决方案目录的“serverhooks”目录中，即可提供函数实施。

在发生特定事件时，除了可由服务器调用这些挂钩外，还可以从您的脚本进行调用。使对这些挂钩的调用同步可避免发生潜在的多线程问题。

启动时，IBM Security Directory Integrator 会在 *serverhooks* 子目录中装入并执行所有用户脚本。脚本可能包含函数声明，也可能不包含。在启动任何配置实例之前，没有函数声明的脚本在启动时将执行一次。定义标准 IBM Security Directory Integrator 服务器挂钩函数的代码以“SDI_”为前缀，操作时将在各个点执行这些代码。

所有 IBM Security Directory Integrator 服务器挂钩函数均具有以下 JavaScript 签名:

```
/**
   * @param NameOfFunction The configuration instance invoking the function
  * @param source The component invoking the function
  * @param user Arbitrary parameter information from the source
  */
 function SDI_functionName(Name_of_function, source, user) {
 }
```

“NameOfFunction”和“source”参数总是分别提供对配置实例和调用组件的访问。“user”参数在各种挂钩函数中用于不同目的。

下列标准函数名由各种 IBM Security Directory Integrator 组件调用:

函数名	调用者（源）	用户参数和期望的值
SDI_ALStarted	配置实例（Config Instance）	当 AssemblyLine 启动时调用。 user = 已启动的 AssemblyLine *返回忽略的值*
SDI_ALStopped	配置实例（Config Instance）	当 AssemblyLine 停止时调用。 user = 已停止的 AssemblyLine *返回忽略的值*
SDI_ConfigStarted	服务器	当配置实例启动时调用。 user = 配置实例 *返回忽略的值*
SDI_ConfigStopped	服务器	在配置实例停止后调用。 user = 配置实例 *返回忽略的值*
SDI_Shutdown	服务器/配置实例	在 IBM Security Directory Integrator 服务器终止 JVM（例如，System.exit()）之前立即调用。 user = 退出状态（整数） *返回忽略的值*

通过 `main.invokeServerHook()` 方法访问 IBM Security Directory Integrator 服务器挂钩函数。同步此函数可防止一次有多个线程执行挂钩。所有调用都是同步调用的，因而调用者将等待函数返回。因此，不应该在服务器挂钩中花费太多的时间。

如先前所述，通过在解决方案目录的“serverhooks”子目录中创建文件来定义脚本并使其可用。在将包含敏感信息的脚本添加到目录中之前，应该使用服务器 API 对它们进行加密。`serverapi/cryptoutils` 工具可用于加密脚本文件。请注意，IBM Security Directory Integrator 会自动尝试解密扩展名为 .jse 的文件，因此加密文件最好应具有该扩展名。

此外，serverhooks 目录中的文件是在首先对平台的标准整理顺序使用区分大小写的排序将文件名排序后装入并执行的。顶级目录中的所有文件都是在处理所有子目录中的文件之前装入的。

服务器挂钩使用的一些示例如下:

- 启动 IBM Security Directory Integrator 时，可通过挂钩到服务器“挂钩”的 JavaScript 片段对始终要装入 IBM Security Directory Integrator 以供脚本编制的定制对象进行实例化操作。这样就向您提供了更多的控制权，而不仅仅是引用“配置浏览器”中 Java 库文件夹下的类。
- 您启动的一个或多个定制 AL 将为这些事件创建审计日志，或使用某些传输方法（SNMP、HTTP 和 JMS 等等）将这些事件传播到其他系统中。
- 每次装入配置或启动 AL 时将调用实施的企业安全策略。

从脚本调用服务器挂钩: com.ibm.di.server.RS 类（脚本变量“main”）具有用于调用服务器挂钩的方法:

```
/**
 * Invokes a server hook.
 *
 * @param name The name of the hook (also the filename)
 * @param caller The object invoking the hook
 * @param userInfo Arbitrary information to the hook from the caller
 */

public Object invokeServerHook(
 String name,
 Object caller,
 Object userInfo) throws Exception;
```

该调用可以返回 Java 对象（任何类型），因此即使 IBM Security Directory Integrator 在执行服务器挂钩期间忽略此调用，您也可以在自己的脚本调用中使用已返回的值。

在 AssemblyLine 内的脚本中，可以执行以下代码:

```
main.invokeServerHook("MyCustomHook", this, "custom information");
```

访问 AssemblyLine 内部的 AL 组件

每个 AL 组件都以具有为该组件选择的名称的预注册脚本变量提供。

请注意，您可以通过对诸如 system.getConnector() 之类的函数进行脚本调用来动态装入组件，但这适用于有经验的用户[8]。

AssemblyLine 参数传递

有三种方式将数据放入 AssemblyLine:

- 在 AssemblyLine 中（例如，在 Prolog 脚本中）生成您自己的初始条目。
- 从一个或多个迭代器供给。
- 从使用 AL 连接器或 AL 函数组件，或使用 API 调用的另一个 AssemblyLine 带参数启动 AssemblyLine。

8. 从该调用中获取的连接器对象是*连接器接口*对象，并且是 AssemblyLine 连接器中特定于数据源的部分。当更改任何连接器的*类型*时，实际上是在换出其数据源智能（连接器接口），该数据源智能提供在特定系统、服务或数据存储器上访问数据的功能。AssemblyLine 连接器的大多数功能（包括属性映射、链接条件和挂钩）由 IBM Security Directory Integrator 内核提供，并在切换连接器类型时保持不变。

如果您想要用来自其他 AssemblyLine 的参数启动 AssemblyLine，则您有两个选择:

- 使用**任务调用块**（TCB），这是首选的方法。TCB 详细描述如下。
- 直接提供“初始工作条目”。

注: 提供这两个选项是为了保持与先前版本的兼容性。

任务调用块（TCB）

任务调用块（TCB）是一种特殊的 Entry 对象，由调用程序用于为 AssemblyLine 设置若干参数。

基本用法: 任务调用块（TCB）是一种特殊的 Entry 对象，由调用程序用于为 AssemblyLine 设置若干参数。TCB 可以为您提供由 AssemblyLine 指定的输入或输出参数的列表（包括在 AssemblyLine 的**操作**选项卡中定义的操作代码），从而使调用程序为 AssemblyLine 的连接器设置参数。TCB 由以下逻辑部分构成:

- 传递至 AssemblyLine 的“初始 Work 条目”: `tcb.setInitalWorkEntry()`
- 连接器参数: `tcb.setConnectorParameter()`
- AssemblyLine 的输入或输出映射规则（在配置编辑器的**操作**选项卡下设置）
- 可选的用户提供的*累加器*对象，它从 AssemblyLine 接收所有工作条目: `tcb.setAccumulator()`

例如，通过以下代码可以用“初始工作条目”启动 AssemblyLine 并将一个名为 MyInput 的连接器的 *filePath* 参数设置为 d:\myinput.txt:

```
var tcb = system.newTCB();                          // Create a new TCB
var myIWE = system.newEntry();                      // Create a new entry object
myIWE.setAttribute("name","John Doe");              // Add an attribute to myIWE
tcb.setInitialWorkEntry ( myIWE );                  // Set the IWE and parameters
// Note that since this is a JavaScript string, we must "escape" the forward slash
//      or use a backslash (Windows syntax)
tcb.setConnectorParameter ( "MyInput", "filePath", "d:\\myinput.txt" );

var al = main.startAL ( "MyAssemblyLine", tcb ); // Start the AL with the tcb
al.join();                                              // Wait for AL to finish
```

启动带有操作的 AssemblyLine: 可以用*操作*定义 AssemblyLine；操作是为 AssemblyLine 定义多个输入映射的概念。根据调用 AssemblyLine 的方式，将激活不同的输入映射。在 AssemblyLine 内部，您将需要检查操作条目以找出活动的操作，并使用“分支组件”来定制从 AssemblyLine 到相关操作的流程。

启动带有操作的 AssemblyLine 的一种方法是用 TCB 和脚本代码。

如果名为“al1”的 AssemblyLine 具有操作“缺省”、“操作 1”和“操作 2”，那么该脚本将启动操作设置为“操作 1”的 AL:

```
var tcb = system.newTCB("al1");
tcb.setALOperation("Op1");
main.startAL(tcb);
```

未指定任何操作将启动操作设置为“缺省”的 AL:

```
var tcb = system.newTCB("al1");
main.startAL(tcb);
```

在 AL 没有“缺省”操作的情况下（例如，仅“操作 1”和“操作 2”），第二个脚本将抛出异常。

关于 AssemblyLine 操作概念的更多信息可在 *参考* 中的“Creating new components using Adapters”一节中找到。

使用累加器: 如先前所提到的，您还可以用 TCB 将累加器对象传递到 AssemblyLine。累加器可以是以下类或接口中的任意一项:

java.util.Collection
克隆所有工作条目并添加至集合（例如 ArrayList、Vector 等）中。

com.ibm.di.connector.ConnectorInterface（连接器接口）
将在每个 AssemblyLine 循环结束时通过工作条目调用此连接器接口的 putEntry() 方法。

com.ibm.di.parser.ParserInterface（解析器）
在每个 AssemblyLine 循环结束时通过工作条目为该解析器调用 writeEntry() 方法。

com.ibm.di.server.AssemblyLine Component（AssemblyLine 连接器）
在每个 AssemblyLine 循环结束时通过工作条目为该 AssemblyLine 连接器调用 add() 方法。

如果累加器不是这些类或接口之一，将返回异常。

例如，要将 AssemblyLine 的所有工作条目累加到 XML 文件中，则可以使用以下脚本:

```
var parser = system.getParser ( "example_name.XML" ); // Get a Parser
// Set it up to write to file
parser.setOutputStream ( new java.io.FileOutputStream ( "d:/accum.xml" ));
parser.initParser();                                  // Initialize it.
tcb.setAccumulator ( parser );                        // Set Parser to tcb

var al = main.startAL ( "MyAssemblyLine", tcb );      // Start AL with tcb
al.join();                                            // Wait for AL to finish

parser.closeParser();    // Close the parser - this flushes and
              // closes the output file
```

同样，您可以配置连接器而不是如以下脚本中那样对解析器手动编程:

```
var connector = system.getConnector("myFileSysConnWithXMLParser");
tcb.setAccumulator ( connector );

var al = main.startAL( "MyAssemblyLine", tcb);
al.join();
connector.terminate();
```

通常，先初始化 TCB，然后 AssemblyLine 将使用此 TCB。如果 AssemblyLine 具有操作规范，则 TCB 将按照 AssemblyLine 的期望将输入属性重新映射到初始工作条目，并同样设置结果对象。这么做是为了使 AssemblyLine 的外部调用接口能够保持相同，即使内部工作条目名称在 AssemblyLine 中更改时也是如此。一旦 TCB 传递至 AssemblyLine，则 TCB 不会再返回任何东西。使用 AssemblyLine 的 `getResult()` 和 `getStats()` 可检索结果对象和统计值。

TCB 结果映射在 Epilog 之前执行，因此您仍然可以在 AssemblyLine 调用程序到达最终结果之前访问最终结果。

禁用 AssemblyLine 组件: 在 IBM Security Directory Integrator 中，可以指定在 AssemblyLine 初始化时不得创建或初始化特定 AssemblyLine 组件。通过使用 TCB 禁用那些组件可以实现该操作。

AssemblyLine 组件在缺省情况下是启用的。

为启用或禁用 AssemblyLine 中的组件，必须对 AssemblyLine 的 TCB 对象调用 com.ibm.di.server.TaskCallBlock.setComponentEnabled(String name, boolean enabled) 方法。该方法的名称参数指定将要启用或禁用的组件名称。方法的启用参数指定组件是要启用还是禁用。

AssemblyLine 组件的实际启用或禁用发生在 com.ibm.di.server.TaskCallBlock.applyALSettings(AssemblyLineConfig alc) 方法中。AssemblyLine 初始化时调用该方法。当进行 AssemblyLine 初始化时，已标记为“禁用”的组件将不会进行创建/初始化。

如果循环组件被禁用，那么所有包含在该循环中的组件也被禁用。

即使从配置编辑器中禁用组件，也可以使用该脚本调用进行启用:

```
com.ibm.di.server.TaskCallBlock.setComponentEnabled(String name, boolean enabled)
```

提供初始工作条目（IWE）

提供初始工作条目（IWE）是使用 TCB 传递参数的备用方法，并因与较早版本兼容而受到支持。

当通过脚本中的 **system.startAL()** 调用启动 AssemblyLine 时，仍可以通过在初始工作条目（可通过 *work* 变量访问）中设置属性或特性值向 AssemblyLine 传递参数。然后由您来应用这些值以设置连接器参数（例如，在 AssemblyLine **Prolog – Init** 挂钩中使用 *connectorName*.setParam() 函数）。

注: 必须使用 task.setWork(null) 调用来清除工作条目，否则 AssemblyLine 中的迭代器会通过第一个循环。

可以使用 getResult() 函数来检查 AssemblyLine 的结果（它是 AssemblyLine 停止时的工作条目）。另见 *参考* 中的“runtime provided Connector”。

下面是在连接器参数值中用 IWE 传递的一个示例:

```
var entry = system.newEntry();
entry.setAttribute ("userNameForLookup", "John Doe");

// Here we start the AssemblyLine
var al = main.startAL ( "EmailLookupAL", entry );

// wait for al to finish
al.join();
var result = al.getResult();

// assume al sets the mail attribute in its working entry
task.logmsg ("Returned email = " + result.getString("mail"));
```

在连接器中编制脚本

输入映射和输出映射

定制属性映射在这些选项卡中执行。当选择属性时，必须选择**高级映射**复选框，并在编辑窗口输入脚本。请记住，在完成所有必要的处理之后，必须将达到的结果值赋予 ret.value，例如:

```
   ...
ret.value = myResultValue;
```

或者，在 IBM Security Directory Integrator 中您必须在要作为结果传递的简单值前使用关键字 *return*:

```
return "mystring";
```

连接器 挂钩

挂钩为您提供响应发生的特定事件的方法，并覆盖连接器的基本功能。在对挂钩进行脚本编制时，您有权访问全局对象，虽然有些标准对象不是在每个挂钩中都可用。有关临时对象可用性的详细信息，请参阅 *参考* 中的“AssemblyLine and Connector mode flowcharts”章节。您还可以完全控制环境、AssemblyLine、连接器、条目以及属性。挂钩为您提供用于定制过程流的各种控制点。请参阅第 27 页的『AssemblyLine 流程和挂钩』。

通过编制脚本设置内部参数

可以使用以下脚本为连接器设置连接参数:

```
myConnector.setParam ( "filePath", "examples/scripting/sample.csv" );
```

这通常是您在 Prolog 中执行的操作，但是当 AssemblyLine 正在运行时它可能很有用（假设您停止并重新初始化连接器）。

```
myConnector.terminate();
myConnector.setParam ( "filePath", "examples/scripting/sample.csv" );
myConnector.initialize(null);
```

在解析器中编制脚本

在解析器中编制脚本实际上是指通过编制脚本实现自己的解析器。该过程的描述包含在 *参考* 中的“脚本解析器”这一章节中。

Java + Script ≠ JavaScript

JavaScript 不是 Java。它可能看起来像 Java，但它实际上只是非常接近，以至于却是引起了混淆。当 Netscape 首次创建它时，JavaScript 最先被称为 *Live!Script* 。虽然 JavaScript 受到了广泛的支持，但是您会了解到还有方言存在；例如，Microsoft 的版本称为 *JScript*。有一个被称为 *ECMAScript* 的标准定义，可以在以下 URL 中找到其规范: http://www.ecma-international.org/

虽然语法相似，但是 Java 和 JavaScript 处理数据和数据类型的方式不同。这是主要混淆源之一，因此在使用 JavaScript 时会出错。

数据表示

Java 自持某些被称为基本的类型，它们时一些简单值，例如有符号的证书、十进制值和单字节。原语不提供任何功能，只提供非复杂数据内容。在计算和表达式中使用它们，将它们分配给变量，以及在函数调用之间传递。Java 还提供大量不携带数据内容（甚至是高度复杂的数据）的对象，但是也提供对象函数（也称为方法）形式的智能。

当执行脚本调用 `task.logmsg( "Hello, World" )` 时，可以调用任务对象的 `logmsg()` 方法。

许多 Java 原语具有对应的对象。一个示例为整数，可以按整数的基本形式 *(int)* 或通过操作 `java.lang.Integer` 对象来对这些整数进行使用。

JavaScript 不使用原语的概念。相反，所有数据都表示为 JavaScript 对象。此外，与 Java 丰富的数据词汇表相比，JavaScript 只具有少量的本机对象。因此 Java 能够区分非小数数字对象和它们的十进制数字对象 – 甚至能够区分有符号类型和无符号类型，并为不同的精确度提供相似的对象 – JavaScript 将所有的数字值都归为一个对象类型，称为 *Number*。

因此，当在 JavaScript 中比较数字值时，可能会得到看似错误的结果，例如:

```
if (myVal == 3) {
 // do something here if myVal equals 3
}
```

如果 myVal 是通过算术运算设置的，或通过引用 Java 十进制对象设置的，则对象的值可以为 3.00001 或 2.99999。尽管这非常接近 3，但是不会通过上面的等值测试。要避免这种特殊问题，可将操作数转换为 Java Integer 对象，以确保得到一个有符号的、非小数值。然后，您的布尔表达式就会按预料得那样执行。

```
if (java.lang.Integer( myVal ) == 3) { ...
```

或者可以确保变量引用了适当的 Java 对象。通常，您将会意识到您所使用的对象的类型。

歧义函数调用

Java 还提供称为 *char* 的基本类型，它可以包含单个字符值。字符的集合可以在 Java 中表示为字符原语的数组，或者可以将其处理为 java.lang.String 对象。正如之前所述，JavaScript 不理解原语。必须使用 JavaScript *字符串*对象来处理字符数据。即使在 JavaScript 中指定了单个字符（″a″），这仍然被视为 *string*。

现在请考虑，当从您的脚本中调用 Java 函数时，该函数会使用函数名称以及用于调用的参数数量和类型来匹配实际的那个方法。这种匹配方式通过将 LiveConnect 扩展到 JavaScript 来执行。LiveConnect 尽力寻找您所引用的函数特征符，这不是一个小任务，因为 Java 和 JavaScript 是以不同的方式表示参数类型的。但是 JavaScript 和 LiveConnect 将会为您做一些内部转换，尝试将 Java 和 JavaScript 数据类型进行匹配。

您有一个方法的多个版本，并且每个版本都有一组不同的参数时，问题就出现了；尤其是，如果两个函数具有相同数量的参数，但是类型不同，**并且**如果这些类型在 JavaScript 中未被区分。让我们看一下一个执行字符串 MD5 加密的示例脚本。

```
// Create MessageDigest object for MD5 hash
var md = new java.security.MessageDigest.getInstance( "MD5" );

// Get the EID attribute value as byte array.
var ab = java.lang.String( "message to encrypt" ).getBytes();

md.update( ab );

var retHash = md.digest();
```

以上的 update() 调用将失败，并会抛出求值异常，表明该函数调用有歧义。这是因为 MessageDigest对象具有该函数的多个版本，每个版本都有相似的特征符: 有一个版本接受单字节，还有一个版本需要字节数组 (byte[])。如果您可以找到同一个方法的另一种变体，它使用不同数量的参数，从而为该方法提供一个唯一可标识的特征符的话，那么您就可以解决这个问题。幸好，MessageDigest 提供了适合的项: 某个版本的 update()，该版本采用字节数组外加一对数字值（偏移量和长度参数）。因此您可以更改代码以转而使用此调用:

```
md.update( ab, 0, ab.length );
```

最后，您可以经常通过将您想使用的 Java 方法括在对象后面的括号中，来指定确切的 Java 方法特征符，具体如下:

```
md["update(byte[])"](ab);
```

这里我们调用的 update()函数版本是以单字节数组参数声明的。

Java 中的字符/字符串数据与 JavaScript 字符串

Java 和 JavaScript 都提供 String 对象。尽管这两种类型的 String 对象工作方式相似，并且都提供了类似的函数，但是它们在很多方面都不相同。例如，每种对象类型对于返回字符串的长度提供了不同的机制。对于 Java 字符串，可以使用 length() 方法。另一方面， JavaScript 字符串具有长度 *变量*。

```
var jStr_1 = new java.lang.String( "Hello, World" ); // Java String
task.logmsg( "the length of jStr_1 is " + jStr_1.length() );

var jsStr_A = "Hello, World"; // JavaScript String
task.logmsg( "the length of jsStr_A is " + jsStr_A.length );
```

这个微小的差别会导致令人困惑的语法错误。尝试调用 jsStr_A.length() 将产生运行时错误，因为该对象没有 length() 方法。

更令人迷惑的错误可能发生在进行字符串比较时。

```
var jsStr_A = "Hello, World"; // JavaScript String
var jsStr_B = "Hello, World"; // JavaScript String

if ( jsStr_A == jsStr_B )
 task.logmsg( "TRUE" );
else
 task.logmsg( "FALSE" );
```

按照设想，您将从上面的片段中得到“TRUE”的结果。但是，与使用 Java 字符串有些许的不同。

```
var jStr_1 = java.lang.String( "Hello, World" ); // Java String
var jStr_2 = java.lang.String( "Hello, World" ); // Java String

if ( jStr_1 == jStr_2 )
 task.logmsg( "TRUE" );
else
 task.logmsg( "FALSE" );
```

这会产生“FALSE”，因为上面的等值运算符将比较两个变量是否都引用内存中的*同一对象* ，而不是匹配它们的值。要比较 Java String 的值，则必须使用相应的 String 方法，具体如下:

```
if ( jStr_1.equals( jStr_2 ) ) ...
```

但是请稍等，还有一个问题。下面的代码片段将使您得到“TRUE”的结果:

```
var jsStr_A = "Hello, World"; // JavaScript String
var jStr_1 = java.lang.String( "Hello, World" ); // Java String

if ( jsStr_A == jStr_1 )
 task.logmsg( "TRUE" );
else
 task.logmsg( "FALSE" );
```

由于 JavaScript 无法操作未知类型，例如 Java String 对象，所以为了执行求值操作，它将首先把 jStr_1 转换为等同的 JavaScript String。

总之，要意识到您所处理的对象的类型。请记住，IBM Security Directory Integrator 函数始终返回 Java 对象。牢记这些因素将有助于将脚本代码中的错误降至最低。

变量作用域和名称

JavaScript 是一种相对非正规的语言，不需要先定义变量然后对它们赋值。它既不强制执行严格类型检查，在重新定义变量时也不会发信号。这使得 JavaScript 可以被快捷方便地使用，但是也很容易导致晦涩难懂的代码和令人困惑的错误，特别是由于您创建了可覆盖内置变量的变量时尤其如此。

有一种难以调试的错误发生在声明了与内置变量名称相同的变量（例如 work、conn 和 current）时，因此您需要熟悉由 IBM Security Directory Integrator 使用的保留名称。

另一种常见的问题发生在您创建的新变量重复定义了现有变量时，这些现有变量可能是在内含的“配置”或“脚本库”中使用的。如果您能够谨慎对待变量名称和它们的*作用域*，则这些错误都可以避免。作用域定义了变量的影响范围，在 IBM Security Directory Integrator 中，我们会谈及全局变量的作用域，那些在所有的挂钩、脚本组件和属性映射中都可用的变量以及那些作用域限于函数本地的变量。

为了更好地理解作用域，必须首先理解每个 AL 都有自己的脚本引擎，并因此运行于各自的脚本环境中。任何变量，如果没有特地定义为函数声明内部的本地变量，那么对于该脚本引擎就都是全局的。因此下面的代码将创建一个全局变量:

```
myVar = "Know thyself";
```

从此刻开始，这个变量将在 AL 的生存期内一直可用。要使这个变量变成本地变量需要两个步骤: 在声明该变量使用 var 关键字，并将该段声明放在函数内部，具体如下:

```
function myFunc() {
 var myVar = "Know thyself";
}
```

现在，如上定义的 myVar 在右花括号之后就将不复存在。请注意，仅仅将变量放置在函数内部还不够必须还要使用 var，以表明您正在声明的是一个新的、本地作用域变量。

```
var glbVar = "This variable has global scope";
glbVar2 = "Another global variable";

function myFunc() {
    var lclVar = "Locally scoped within this block";
    glbVar3 = "This is global, since "var" was not used";
};
```

只要在函数内部声明了本地作用域的变量，则可以随意对其进行调用。一旦出了作用域，那么任何先前的类型和值都将恢复。

尽管对于定义全局变量而言，var 关键字并不是必需的，但是最好让然应该这么做。建议在您的脚本开头部分定义变量，包括能使读者理解这些变量的用途的充分注释。该方法不仅能够提高代码的易读性，还会使您对变量的命名和作用域作出明智决定。[9]

9. 函数命名的运作会有所不同。Java 之类的编程语言通过函数名称和数量以及参数的类型来识别函数。JavaScript 只使用名称。因此，如果具有同一函数的多个定义，JavaScript 将只“记住”最后一个函数定义 – 无论定义是否具有不同数量的参数。

实例化 Java 类

可以调用外部 Java 类；即您已使用脚本代码将其添加到 IBM Security Directory Integrator 运行时环境或 CLASSPATH 中的那些类。

例如，假定您想要使用标准 java.io.FileReader，请使用以下脚本:

```
var javafile = new java.io.FileReader ( "myfile" );
```

现在，您拥有称为 *javafile* 的对象；使用此对象，您就可以调用对象的所有方法。

相同的技术将用于您自己对象的实例化，具体如下:

```
var myfile = new my.FileReader("myfile");
```

在脚本编制中使用二进制值

可以通过使用条目的 getObject() 函数从属性检索二进制值。二进制 Attribute 值本身将返回字节数组。这里是一个 JavaScript 示例:

```
var x = conn.getObject("objectGUID");
for ( i = 0; i < x.length; i++ )
{
   task.logmsg ("GUID[" + i + "]: " + x[i]);
}
```

该示例将在 -128 到 127 之间变动的某些数字写入日志文件。您可能希望对数据执行一些其他操作。如果您已从连接器读取了以字节数组形式保存的密码，则可以使用以下代码将其转换称字符串:

```
password = system.arrayToString(conn.getObject("userpassword"));
```

在脚本编制中使用日期值

在 IBM Security Directory Integrator 中处理日期时，意味着使用 java.util.Date 的实例。使用任意可用的脚本编制语言，则可以实施您自己的机制来处理日期；但这不是一个常见的方法。

IBM Security Directory Integrator 脚本编制引擎为您提供用于分析日期的机制。*系统*对象具有可以随时访问的 parseDate(date, format) 方法。

注: 获取 java.util.Date 的实例时，可以使用标准的 Java 库和类来扩展处理。

这里是处理日期的一个简单 JavaScript 示例。可以在任何脚本编制控制点放置和启动此代码:

```
var string_date1 = "07.09.1978";
var date1 = system.parseDate(string_date1, "dd.MM.yyyy");

var string_date2 = "1977.02.01";
var date2 = system.parseDate(string_date2, "yyyy.dd.MM");

task.logmsg(date1 + " is before " + date2 + ": " +
  date1.before(date2));
```

此脚本代码首先将两个日期值（使用不同格式）解析成 java.util.Date。然后使用标准的 java.util.Date**.before()** 方法来确定第一个日期实例是否在第二个实例之前出现。该脚本的输出随后打印到日志文件。

在脚本编制中使用浮点值

以下示例演示在创建的脚本编制代码中如何使用浮点值。所有这些示例都是在 JavaScript 中实现的。虽然使用其他几个脚本语言可以重复相同的示例，但语法可能会不同。以下的简单脚本将浮点值赋予两个变量以找到它们的平均值。可以从任何脚本编制控制点启动此代码。日志文件输出为"r = 3.85"。

```
var a = 5.5;
var b = 2.2;
var r = (a + b) / 2;
task.logmsg("r = " + r);
```

下一个示例扩展这个简单的脚本。假设在您的输入连接器中有一个称作“Marks”的多值属性，它包含表示浮点值的字符串值（java.lang.String）。这是常见情况。此属性映射至输出连接器中称作“AverageMark”的属性，它保存“Marks”属性所有值的平均值。以下代码用于“AverageMark”属性的高级映射中:

```
// First return the values of the "Marks" attribute
var values = work.getAttribute("Marks").getValues();

// Zero out counter and sum variables
var sum = 0;
var count = 0;

// Loop through the values, counting and summing them
for (i=0;  i<values.length; i++)
{
   // use the Double() function to convert value to number
   sum = sum + new Number(java.lang.Double(values[i]));
   count++;
}

// If count > 0, compute the average
var average = (count > 0) ? (sum / count) : 0;

// Return the computed average
ret.value = average;
```

本示例中的中心调用就是 java.lang.Double(values[i])，它用于将当前已建立索引的“Marks”值转换为之后可在平均值计算中使用的数字值。

第 3 章 配置编辑器

IBM Security Directory Integrator Eclipse 配置编辑器 (CE) 是用于开发 IBM Security Directory Integrator 解决方案的主要工具。通过它您可以创建、维护、测试和调试配置文件；它构建在 Eclipse 平台上，提供了既全面又可扩展的开发环境。

为了更容易理解此处出现的概念，您应该熟悉常见的 Eclipse 概念，如编辑器、视图和其他常见的 Eclipse 扩展点。

项目模型

随着基于 Eclipse 的配置编辑器 (CE) 的出现，IBM Security Directory Integrator (IBM Security Directory Integrator) 中解决方案的开发将不基于纯配置文件的开发（如 V7.0 版本之前一样），而是基于项目模型和*工作空间*。您可以使用项目模型和工作空间进行开发工作；准备部署解决方案时，您可以对工作空间中的配置文件进行解压缩，然后将其发送至适当的 IBM Security Directory Integrator Server: 运行时环境。这有几种好处，其中一些好处包括:

- 配置文件更干净；不会从工作空间导出不必要的和不使用的配置元素;
- 编辑 IBM Security Directory Integrator 配置更高效;
- 更大程度地复用公共组件;
- 能够使用源代码管理系统，例如 CVS。

工作空间

当启动 CE 时，将提示您选择工作空间目录。工作空间目录是 CE 存储所有 IBM Security Directory Integrator 项目和文件的地方。这与解决方案目录不同，尽管解决方案目录内可以很好地包含工作空间目录。CE 从工作空间收集文件，从而形成可在 IBM Security Directory Integrator 服务器上执行的运行时配置文件 (`rs.xml`)。可以认为工作空间内的文件和项目是各种运行时配置文件的源。

安装目录、解决方案目录和工作目录

播放中使用三个不同的目录可能会产生一些混淆。正如在先前部分中所说明的，工作空间目录属于存储项目文件的 CE。运行 IBM Security Directory Integrator 服务器时，安装目录和解决方案目录属于该服务器。但是，当在 CE 中配置连接器时（举个例子），您将经常使用发现属性函数；这将导致 CE 打开连接器并对连接器执行操作。由于一些连接器将文件名用于多个目的，因此当使用相对路径时，可能会混淆。由于这个原因，CE 的缺省工作目录应该一直与主要使用的解决方案目录相同[10]。首先，由于在 CE 内部创建几个服务器和解决方案目录是可能的。您仍可以使用这些解决方案和服务器，并在那些服务器上运行 AssemblyLine，但是当您在 CE 中使用连接器而不是在运行 AssemblyLine 期间使用连接器时，配置中的相对路径不再相同。

10. 工作目录不应该与工作空间混淆。工作空间是存储项目文件的区域；工作目录是文件系统中定义所有非标准文件名的位置。由于可移植性原因，这应该是解决方案目录。

考虑带有文件连接器（使用 abc.txt 作为其输入文件）的配置。当在 CE 中使用发现属性时，它将在 CE 的工作目录（安装时指定的首选解决方案目录）中查找文件。当运行带有该连接器的 AssemblyLine 时，一切似乎都进展顺利。然后创建一个新服务器，其中的解决方案目录指向其他某个位置而不是当前工作目录。接着配置连接器，并且连接器将文件名解析到正确的地方（即当前工作目录中 abc 文件所在的位置）。但当您运行 AssemblyLine 时，它将失败，因为无法找到该文件。这是因为新服务器在不同的工作目录（解决方案目录）中运行而不是在 CE 中运行。

在缺省情况下，CE 的工作目录设置为用于执行 AssemblyLine 的缺省服务器的解决方案目录。因此，如果未作任何更改，那么除了创建新解决方案目录时或更改缺省服务器的解决方案目录时，您不应该看见这些问题中的任何一个。

工作台

工作台是可执行 IBM Security Directory Integrator 解决方案的所有配置和开发的主要用户界面应用程序。工作台由一些视图和使您能够执行这些操作的编辑器构成。

“IBM Security Directory Integrator 服务器”视图

IBM Security Directory Integrator 服务器实例不会在每次运行 AssemblyLine 时都启动，这与早期版本（7.0 之前的版本）不同。IBM Security Directory Integrator 服务器将改为在启动 CE 时自动启动，用于测试和运行开发的 AssemblyLine。与早期版本相反，对象的测试运行完成时，此服务器不会终止，而是保持运行，以等待您的下一次测试运行。

在服务器视图中，您可以管理服务器定义。此处定义的服务器可以是本地服务器或是远程服务器，并且可供创建的各种 IBM Security Directory Integrator 项目使用。定义了安装路径的服务器视为可以由 CE 启动的“本地”服务器。

自动定义了一个名为 *Default* 的服务器。该服务器是新建的 IBM Security Directory Integrator 项目所用的缺省服务器。使用用户定义的解决方案目录根据全局属性文件中的值创建缺省服务器（例如，根据 SDI_SOLDIR 变量，由安装程序依次安装）。

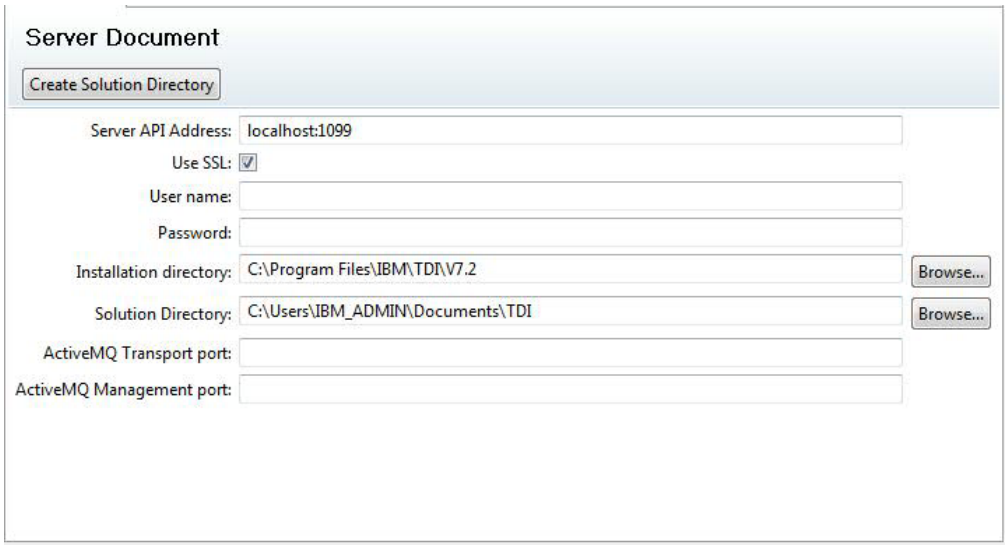


图 5. 缺省 IBM Security Directory Integrator 服务器定义

IBM Security Directory Integrator 项目

要开发 IBM Security Directory Integrator 解决方案，必须先创建一个 IBM Security Directory Integrator 项目。 Eclipse 中的项目是相关文件和资源的集合。

创建项目时，项目中将填充常见的配置对象所在的几个文件夹。如下显示了新建的 IBM Security Directory Integrator 项目的布局。

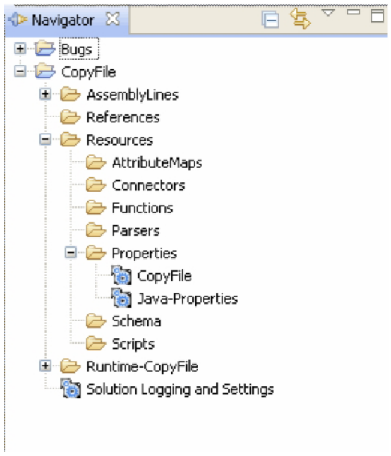


图 6. IBM Security Directory Integrator 项目树

AssemblyLine 文件夹包含项目的 AssemblyLine；而资源文件夹包含 AssemblyLine 共享或使用，或者通常应用于解决方案的所有组件：特性、记录参数等。要创建新资源，可使用主菜单中的**文件/新建...** 向导，或使用每个文件夹中的上下文菜单（也就是，右键单击 AssemblyLine 文件夹以创建新的 AssemblyLine）。

运行时目录（运行时-*项目名称*）是一个包含运行时配置文件和定制属性文件的特殊目录。每当更改项目中的组件（连接器、AssemblyLine、属性文件等）时，此目录中的

相关文件也会更新。所有生成的文件都标记为派生文件，这表示如果您试图修改内容，将会被警告。请注意，如果您确实对这些文件中的任何文件进行更改，那么它们将会被覆盖。运行时目录的目的是创建项目的运行时文件目录，如果使用了源控制，并且可供 IBM Security Directory Integrator 服务器使用，那么可以对该目录进行复制或检出。

配置文件

由服务器运行的配置文件（Config）是一个组合 XML 文档，其中 AssemblyLine、连接器等都是同一文档的一部分。在 IBM Security Directory Integrator Eclipse CE 中，对这些组件中的每个组件都分配了它们自己的物理文件。这些文件仅包含一个配置对象。

在解决方案开发期间将配置文件分割为独立文件的原因之一是使组件共享更容易。同时，使每个组件位于自己的文件中可使源控制系统（如 CVS）和多用户开发（其中不同的人员同时在解决方案的不同部分工作）更好地使用这些组件。

可以使用导入向导来导入此版本之前的配置。这样处理的结果是将导入的配置分割成单独的配置文件。另一种方法是使用**文件 > 打开 Security Directory Integrator 配置文件**选项，该操作会将 7.0 之前的配置导入新建项目中。但是请注意此选项也会重新为源文件设置自动更新。请参阅下一节了解关于链接文件功能的更多信息。

运行时配置文件

运行时配置文件在标准视图中是隐藏的。这是 IBM Security Directory Integrator 服务器用于运行解决方案的文件，也是先前版本（7.0 之前）可直接处理的文件。每当在 CE 中保存配置文件时，也会对运行时配置文件进行更新。该文件也是传送给 IBM Security Directory Integrator 服务器以供执行的文件。此文件由项目构建者维护，最终用户不应该对它进行修改。

您可以配置您的项目以在运行时配置文件更改时自动将其导出。使用项目属性面板可配置项目链接到的文件。

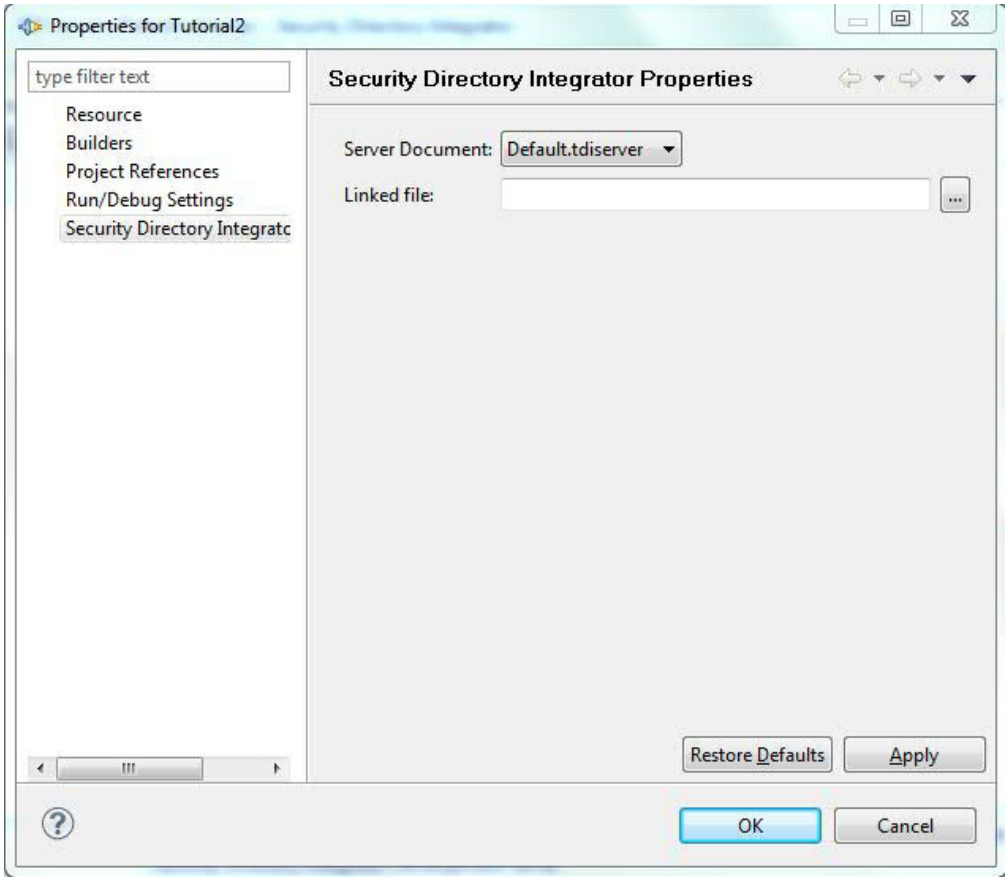


图 7. "IBM Security Directory Integrator 项目属性"窗口

每当更新运行时配置时，也会将其复制到项目的 IBM Security Directory Integrator 属性部分所指定的文件。当您使用**文件 > 打开 Directory Integrator 配置**命令时，会自动设置链接文件。

项目构建器

定制项目构建器与用于将所有工件聚集至*可运行的配置文件*中的项目关联。

每次资源发生更改时都将自动运行该构建器，或者可以通过标准**项目 > 构建**菜单项手动运行该构建器。无论使用上述哪种方法，IBM Security Directory Integrator 项目构建器都会维护调用时进行更新的配置文件。当更改（修改、添加或除去）资源时，构建器将用该资源更新来更新可运行的配置文件。该操作只对已识别的配置文件起作用；例如 AssemblyLine，但对 .gif 文件不起作用。可运行的配置文件经常是隐藏的，因为文件名称以点（.）开头。文件名为 `.rs.xml`，位于项目文件夹下。这是发送给 IBM Security Directory Integrator 服务器来执行的已编译配置文件。

构建器将重新放置并重命名目标配置中的组件。如果 Resources 文件夹包含特性和连接器，那么会将它们重新放置到目标配置中的标准文件夹（即 Properties 和 Connectors 文件夹）中。

项目构建器还将检查所有已修改的组件以查找明显的错误和潜在的问题。将这些问题被记录到标准的 Eclipse **问题**视图中。每个问题项包含问题的描述以及位置，以便您可以在标识问题的地方双击并激活编辑器。

特性和替换

每个 IBM Security Directory Integrator 项目都与 IBM Security Directory Integrator 服务器相关联。关联的服务器是将用于执行项目中的 AssemblyLine 的服务器。由于服务器可能位于不同的机器上，因此项目模型必须提供通过选定特性存储的本地副本访问特性的方法。

属性存储可根据需要从服务器下载，而服务器中的本地副本包含每个属性的两个值。一个值是从服务器下载的内容（远程值），而另一个值是由用户设置的值（本地值）。这样做是为了可以查看可能的冲突中涉及哪些属性，并可以抽取解决方案正在使用的属性集。在将属性添加到属性存储之前，不需要下载属性存储；但是，当运行解决方案时，将针对服务器的值检查带有本地值的任何属性，以避免无意识地覆盖现有属性。

通过打开（或创建）Resources 文件夹中的属性文件可以完成属性文件的编辑。在创建属性文件后，您可以修改其内容并执行上载和下载。上载和下载是将本地的属性与服务器上的属性同步所执行的操作。

注: IBM Security Directory Integrator 当前使用等号“=”或冒号“:”作为键/值对属性文件中的分隔符，以先用的符号为准。因此，不支持在属性名和属性值中使用等号或冒号。 IBM Security Directory Integrator V6.0 和较早版本中的属性文件键/值分隔符只能是“:”符号；因此可能需要对从 V6.0 和较早版本迁移而来的属性文件进行编辑。

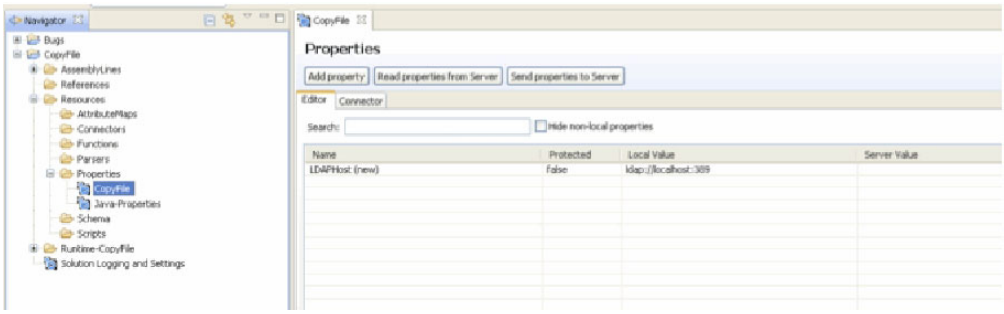


图 8. 属性视图

请注意，带有相关路径的定制特性存储（例如，使用相对路径的连接器配置）将有在运行时 *Project* 目录中生成的相应文件。当创建新定制属性时，缺省的路径将设置为“{config.$directory}/*Filename*.properties”，其中 *Filename* 是您为新属性存储指定的名称。“{config.$directory}”解析为运行时文件的位置。

在缺省情况下，未将共享的属性存储添加到项目中（例如，全局、解决方案和系统存储）。如果想要维持对那些文件的更改，那么您仍可以将这些属性存储添加到项目中。要查看共享的属性存储，您应该使用“服务器”视图或使用主工具栏上的**浏览系统存储**。

用户界面模型

用户界面（UI）是通过一组视图、编辑器和其他 UI 相关的资源提供的。这些视图、编辑器和资源是通过 Eclipse 平台定义的标准扩展点机制提供的。

虽然在 CE 中有很多扩展点添加项，但是这里只列出了最重要的添加项。

表 7. Eclipse CE 扩展点添加项

添加项	描述
编辑器	提供了所有主要配置文件的编辑器: • AssemblyLine（`.assemblyline` 文件） • 连接器（`.connector` 文件） • 函数（`.function` 文件） • 脚本（`.script` 文件） • 特性（`.tdiproperties` 文件） • 属性映射（`.attributemap` 文件）
视图	提供了几个视图以帮助对配置文件的各个方面进行可视化。
菜单，工具栏	CE 中对配置对象执行的所有操作都定义为标准操作。
向导	提供了几个向导以帮助创建新项目和配置文件。

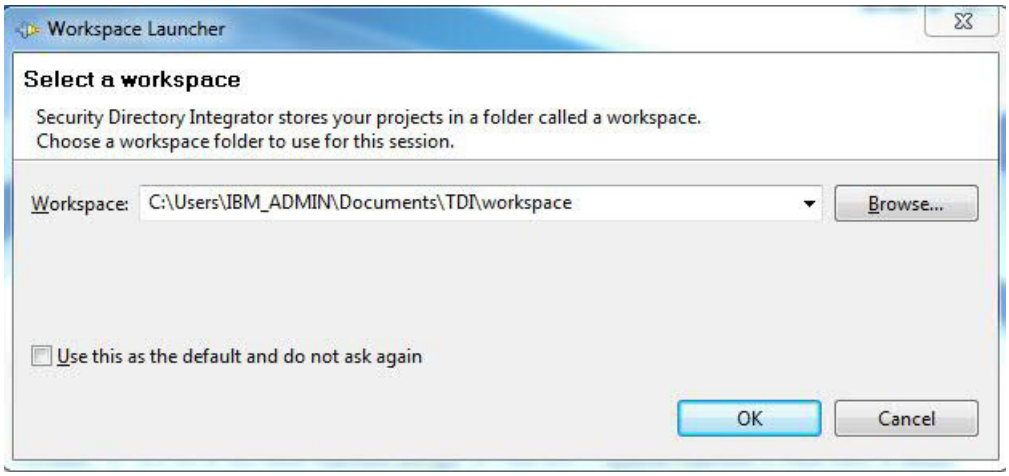
CE 遵循 MVC（模型、视图、控制器）范式。配置文件的编辑器创建显示配置文件内容的窗口小部件。窗口小部件记录与 Eclipse 框架一起使用的工具栏和菜单，以便可以对它们执行添加操作。影响配置文件的所有操作都是使用标准 Eclipse 机制实施和提供的。

用户界面

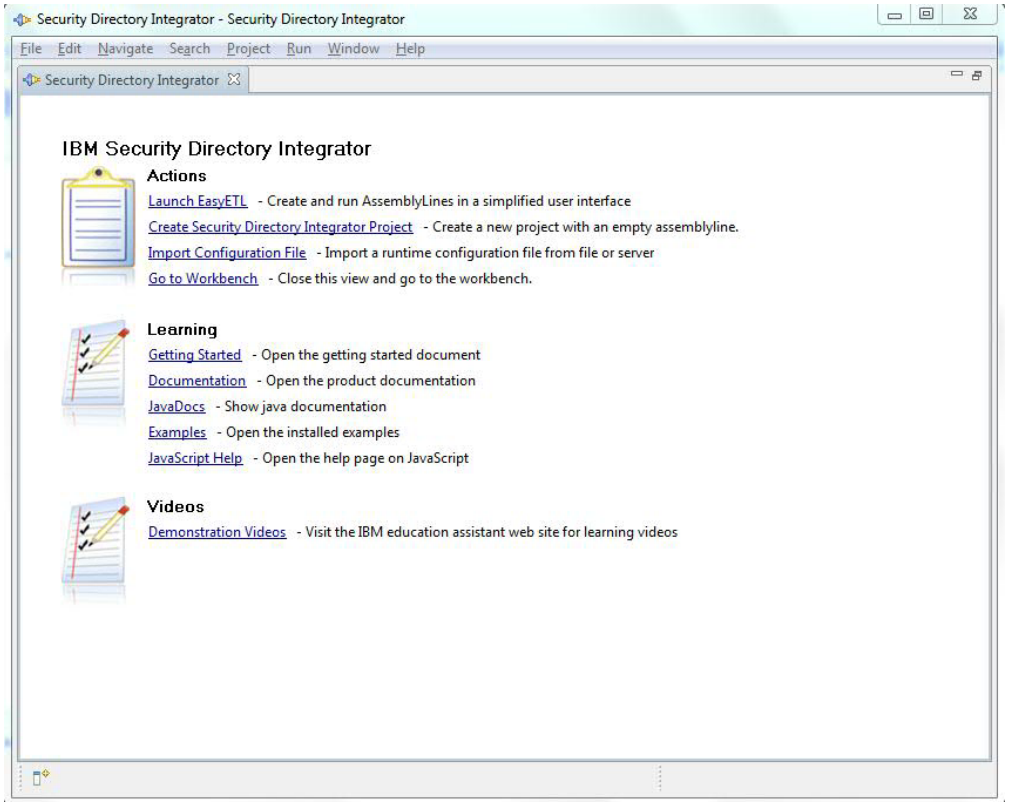
应用程序窗口

首次启动配置编辑器（CE）时，将提示您提供工作空间目录。该工作空间目录是存储项目的文件系统位置。稍后可以从**文件**菜单中更改该目录。

每次启动 CE 时都将显示该对话框，除非您选中该复选框以使指定的工作空间目录作为缺省位置。

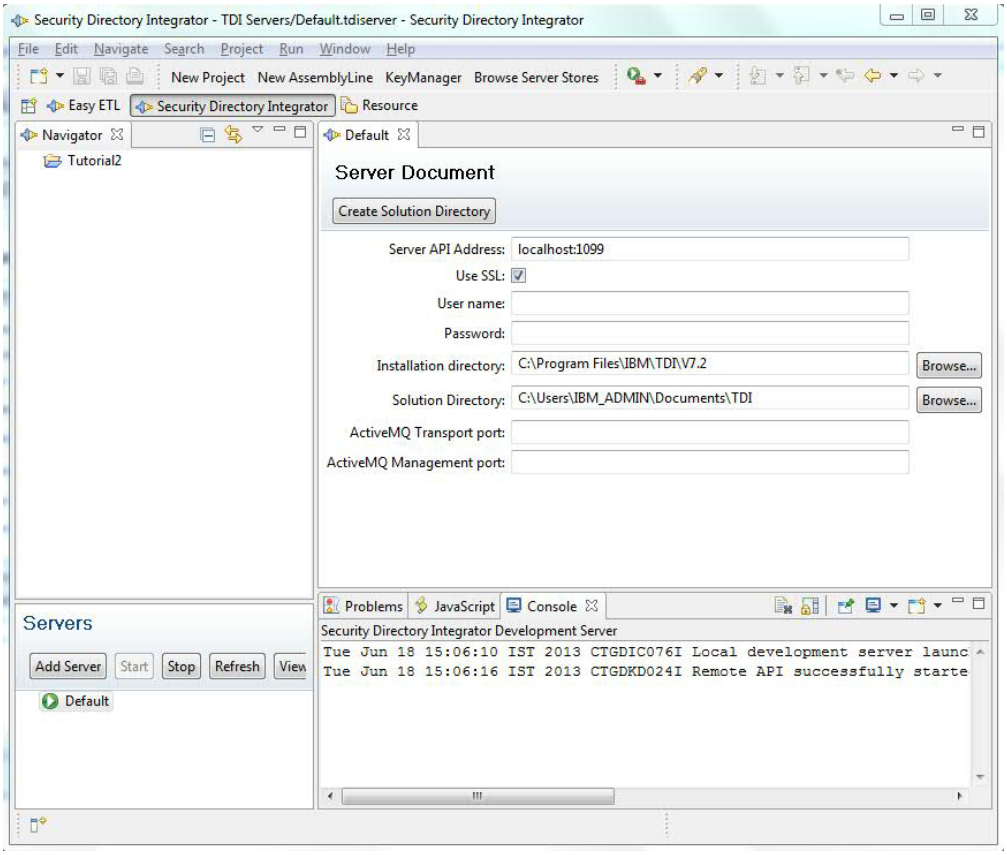


在该对话框之后将显示主工作空间窗口。如果这是您首次启动 CE，将显示欢迎屏幕:



此欢迎屏幕提供了常见任务和信息站点的若干快速链接。文档链接带您查看已配置的产品文档（系统属性 *com.ibm.tdi.helpLoc*）。

以后通过选择**帮助** > **欢迎**，可以稍后重新打开欢迎屏幕。当欢迎屏幕关闭时将看到工作空间窗口:



这是管理项目和配置的主窗口。

在此图片中，有一个打开的编辑器（对于 Default.server 文档）和大量视图。您在此图片中看到的是最重要的视图。

导航器（左上角）包含服务器配置和 IBM Security Directory Integrator 解决方案的所有项目和源文件。该导航器还可以包含其他文件和项目，例如文本文件等。CE 将对 IBM Security Directory Integrator 项目进行专门处理，因此，其他文件和项目不会受到 CE 的影响。您将在该部分中看到有关项目构建器的方式。

*服务器*视图（左下角）将显示“IBM Security Directory Integrator Server”项目中所定义的每个服务器的状态。可以按您的需要定义尽可能多的服务器。该视图提供了要在服务器及其配置上运行的一组功能。刷新按钮将刷新视图中所有服务器的状态。

编辑器区域（右上角）是显示所有编辑器的位置。打开文档时（例如 AssemblyLine 配置），文档将在该区域中结束。此区域被一个包含用来提供其他相关信息的不同视图的区域（右下角）垂直分割。其中最重要的部分是“问题”视图（用于显示 IBM Security Directory Integrator 组件的潜在问题）、“错误日志”（用于显示开发解决方案时发生的错误）、以及“控制台”视图（用于显示运行 IBM Security Directory Integrator 服务器（例如，由 CE 启动的服务器）的控制台日志）。

服务器视图

服务器视图包含了一些用于管理服务器实例的有用功能。除了添加和除去服务器实例之外，还有许多与服务器及其活动配置实例和 AssemblyLine 关联的命令。

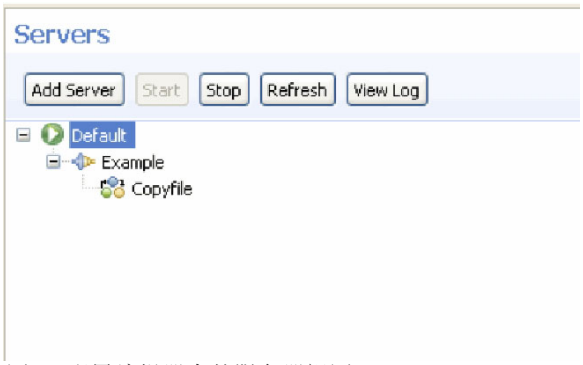


图 9. 配置编辑器中的服务器视图

主工具栏显示以下按钮:

添加服务器
使用此按钮可添加另一个服务器

启动 使用此按钮可以启动“本地”服务器（例如，在您的文件系统中可访问的服务器）。

如果已选择配置实例，那么可以启动其一个或多个 AssemblyLine。

停止 使用此按钮可以向服务器、配置实例或 AssemblyLine 发送停止请求。

刷新 使用此按钮可刷新服务器视图的内容

查看日志
使用此按钮可查看“本地”服务器上的标准日志文件“ibmdi.log”

视图中每项的弹出菜单显示可根据选择执行的其他命令。

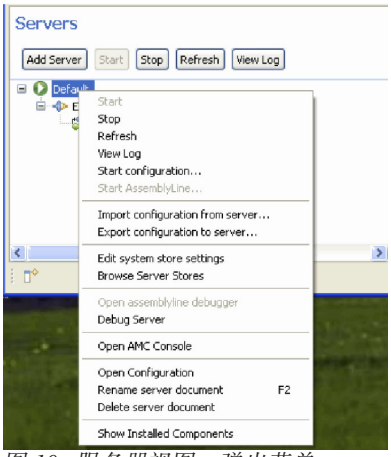


图 10. 服务器视图；弹出菜单

可能的命令为:

启动 启动服务器

停止 停止服务器、配置实例或 AssemblyLine。

刷新 刷新服务器视图。

查看日志
在文本编辑器中打开 ibmdi.log 文件

启动配置...
选择了服务器后，可以在该服务器上启动配置实例。

启动 AssemblyLine...
选择了配置实例后，可以从该配置实例启动 AssemblyLine。

从服务器导入配置...
使您能够从服务器将配置导入到 CE 项目。

将配置导出到服务器...
使您能够将 CE 项目导出到所选服务器上的运行时配置。

编辑系统存储设置
打开选定服务器的系统存储设置

浏览系统存储
打开系统存储数据浏览器以查看/编辑系统存储表。

打开 AssemblyLine 调试器
将调试器连接到选定的 AssemblyLine。这将让您调试已在运行的 AssemblyLine。

调试服务器
打开选定服务器的服务器调试会话

显示已安装的组件
显示选定服务器的已安装组件及版本的列表

打开 AMC 控制台
打开选定服务器的 AMC 控制台。如果服务器已安装但不带 AMC，那么此选项将变灰。

注: 不推荐使用 AMC 功能部件，IBM Security Directory Integrator 的后续版本中将移除该功能部件。

删除服务器文档
从服务器视图删除服务器

重命名服务器文档
重命名选定的服务器。这对服务器本身无效；它仅是服务器的本地表示。

根据选择来禁用和启用菜单选项。

表达式编辑器

表达式编辑器在许多不同的上下文中均可用。通常，在指定参数值（例如：连接参数、链接条件等）时，您可以使用表达式编辑器，而不是为此参数输入一个简单值。

使用属性

此选项使您能够从属性存储中选择现有属性，或者创建新的属性/值对。

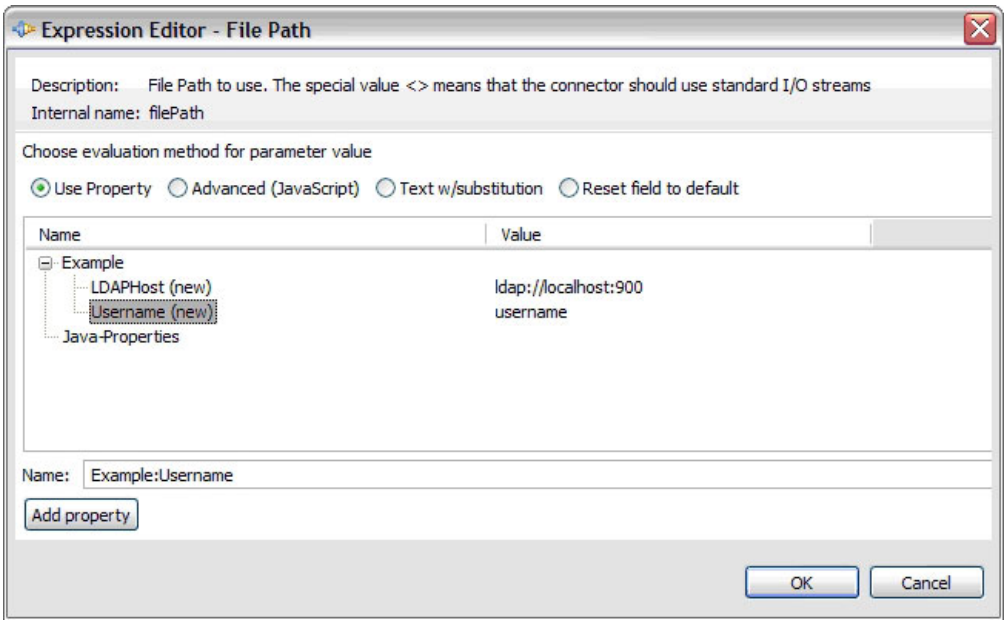


图 11. 表达式编辑器: 简单属性

选择属性时，以此属性的表达式来更新树下的 *Name* 文本字段。缺省情况下，表达式包含存储名称（在此情况下为“Example”），随后是冒号和属性名（在此情况下为“Username”）。单击**确定**后，将对参数使用文本字段中的表达式。这也意味着您可以在此字段中直接输入表达式，而不必通过属性树。例如，如果您知道将运行此解决方案的服务器上有个名为“FilePath”的属性，那么可以除去存储名称（如果您不知道此名称）（即: 输入“FilePath”而不必输入“store:”前缀）。

高级 (JavaScript)

使用此选项，会将值计算为您在编辑器中输入的 JavaScript 代码的结果。

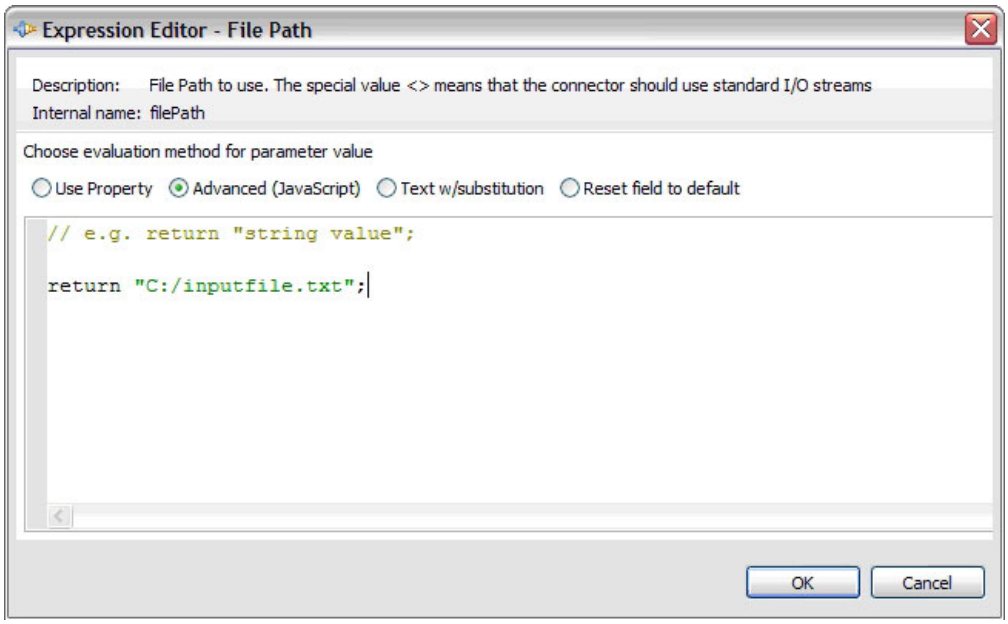


图 12. 表达式编辑器: 高级 (JavaScript)

具有替换的文本

这是 V6.x 中的“IBM Security Directory Integrator 表达式”。从 V7 开始，建议使用 JavaScript 对变量和属性进行复杂求值。但是，在您要输入大量文本时，此选项非常有用。替换选项与 v6 表达式兼容。

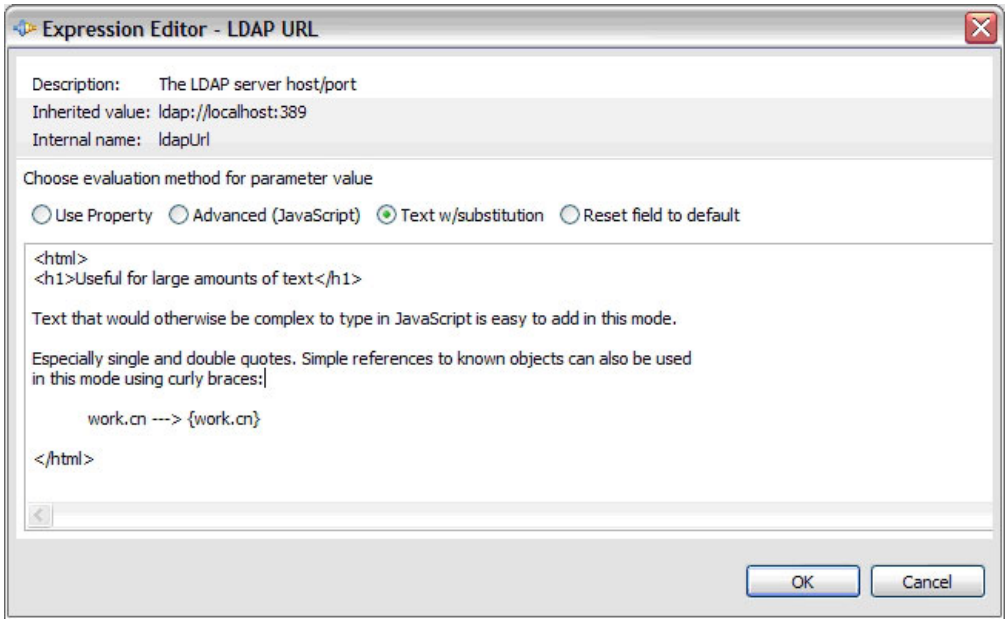


图 13. 表达式编辑器: 具有 v.6 样式替换的文本

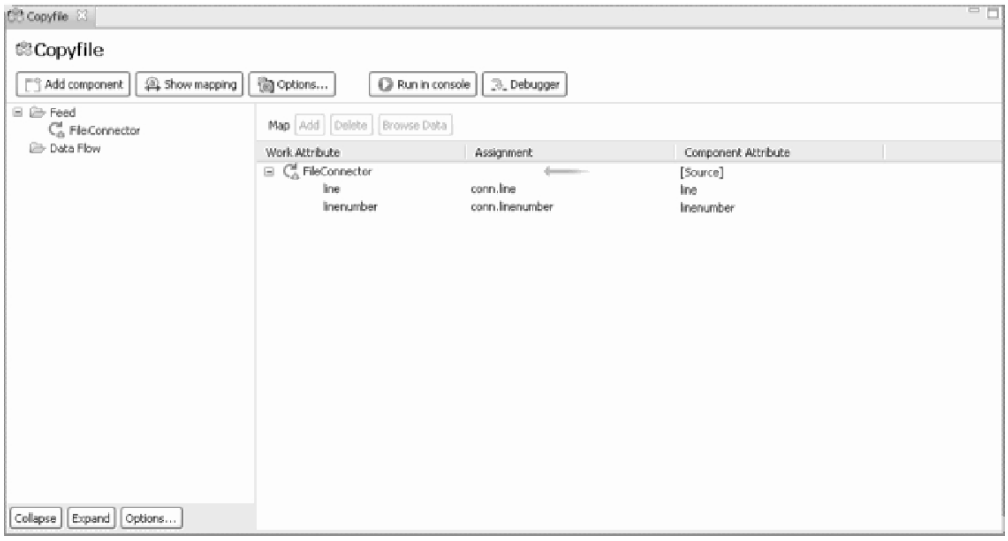
将字段复位为缺省值

将使用最后一个选项来将参数值复位为其缺省值（也称为继承值）。选择此选项并按**确定**来复位参数值。

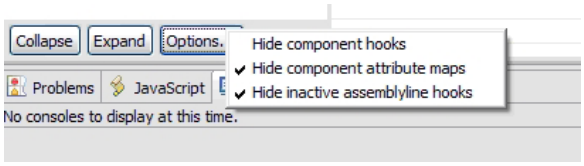
AssemblyLine 编辑器

AssemblyLine 编辑器是开发 IBM Security Directory Integrator 解决方案时使用的主要编辑器。

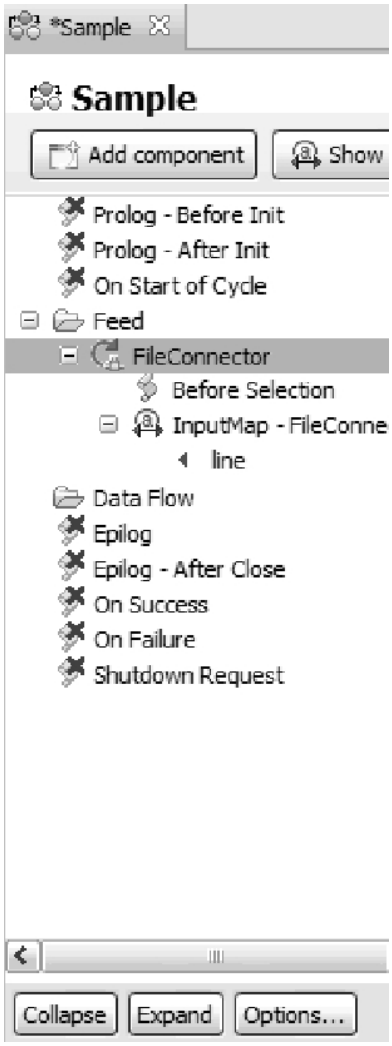
在该编辑器中，通过添加和配置组件来构建 AssemblyLine。在构建 AssemblyLine 的同时，可以运行该 AssemblyLine 以查看已添加组件的效果。



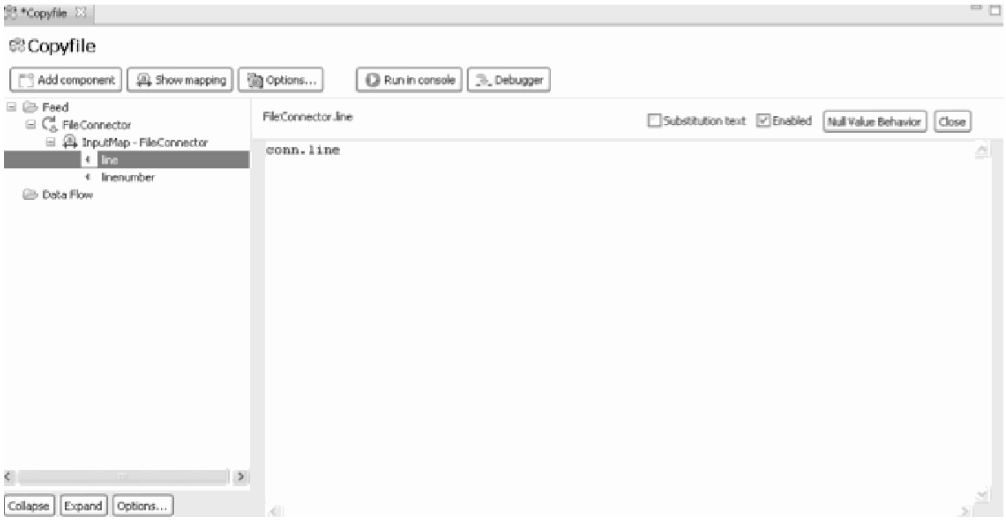
AssemblyLine 编辑器显示两个主要的部分。左侧的部分显示 AssemblyLine 组件和挂钩。在工具栏中，可以选择想要在树中看到的详细级别。



选项... 按钮使您能够选择要在组件树形视图中显示 AssemblyLine 的比例。

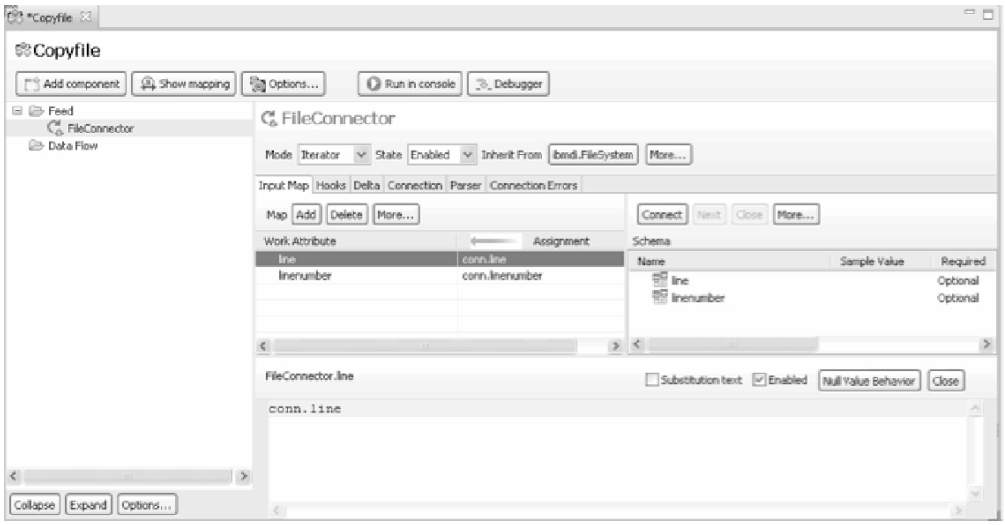


该图片显示树中所有的 AssemblyLine 挂钩以及组件属性映射。在该树中选择挂钩或属性映射项时，右侧将提供比组件编辑器更大的项视图:



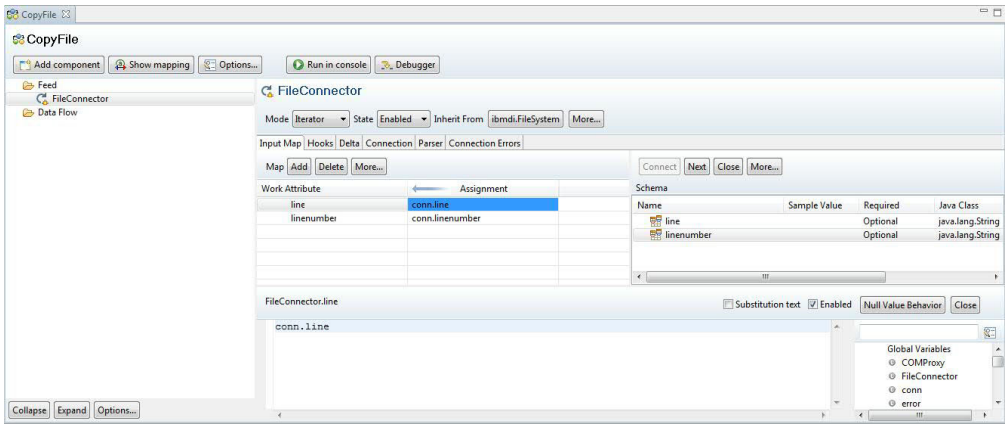
右侧的映射部分显示具有属性映射的所有组件的映射。该树中的第一级是组件名称，其下是各个属性映射项。选择这样的项将启动该项的详细信息编辑器。如果在 AssemblyLine 组件视图中显示挂钩，也可以双击这些挂钩以启动脚本编辑器。

下图显示了快速编辑器如何与属性映射的脚本一起显示。

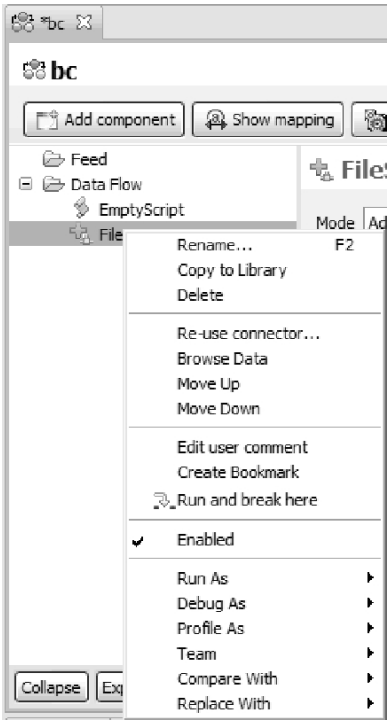


快速编辑器还显示某些其他选项。“替换文本”为 V6“IBM Security Directory Integrator 表达式”格式，您可以在其中输入文本和某些简单的扩展宏。对于 V7，由于 JavaScript 提供更强大的表达式语法，该格式将主要用于较大或复杂的文本值。

组件流部分显示 AssemblyLine 中的所有组件。选择组件时，右侧的编辑器将替换为该组件的配置屏幕。CE 中一个有用的快捷键是 Ctrl-M 快捷键，该快捷键可以最大化当前编辑器或视图以占满应用程序屏幕。Ctrl-M 将在最大化和标准大小之间切换。



单击组件时，该组件的配置屏幕将替换整体属性映射视图。还可以右键单击组件以访问组件的弹出菜单。



AssemblyLine 选项

从**设置**下拉按钮中，可以选择大量选项。

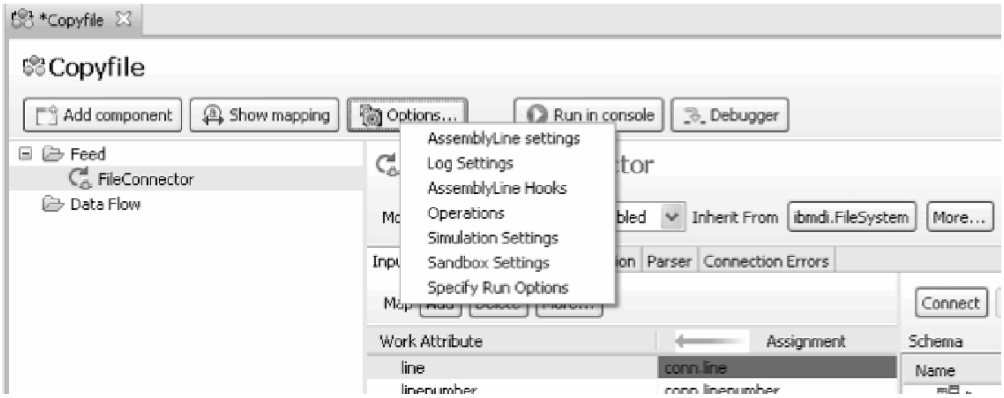


图 14. AssemblyLine 选项菜单

下拉菜单中提供多个选项屏幕来通过设计以及运行时选项对 AssemblyLine 的各方面进行管理。这些屏幕是:

- 第 84 页的『AssemblyLine 设置』
- 第 84 页的『日志设置』
- 第 85 页的『AssemblyLine 挂钩』
- 第 86 页的『AssemblyLine 操作』
- 第 87 页的『仿真设置』
- 第 88 页的『沙箱设置』

AssemblyLine 设置

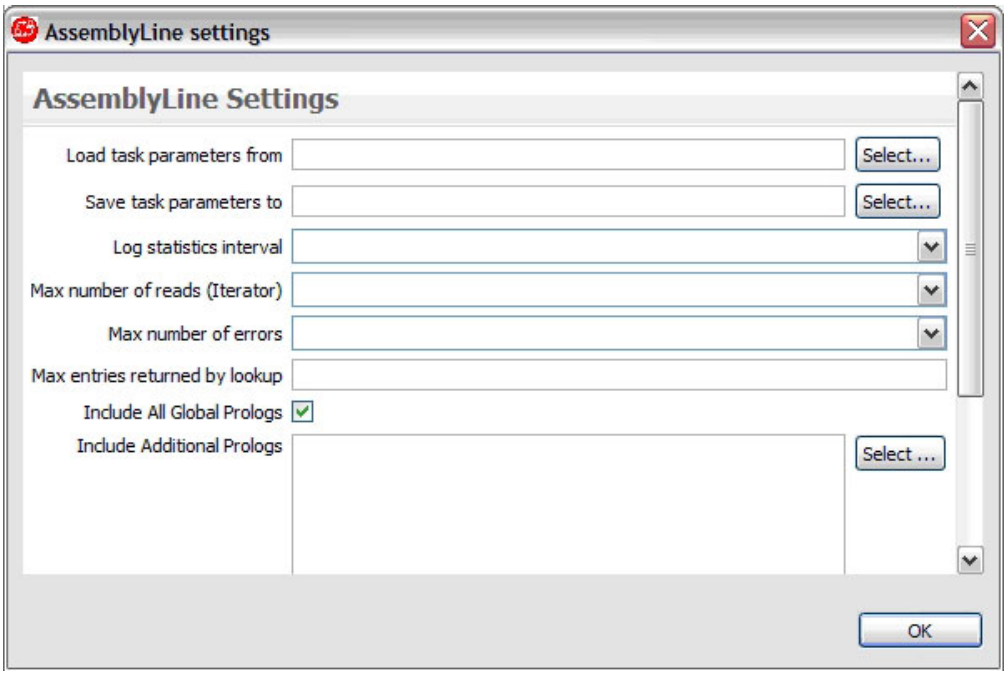


图 15. AssemblyLine 设置

在该窗口中，可以指定影响 AssemblyLine 执行方式的选项。

日志设置

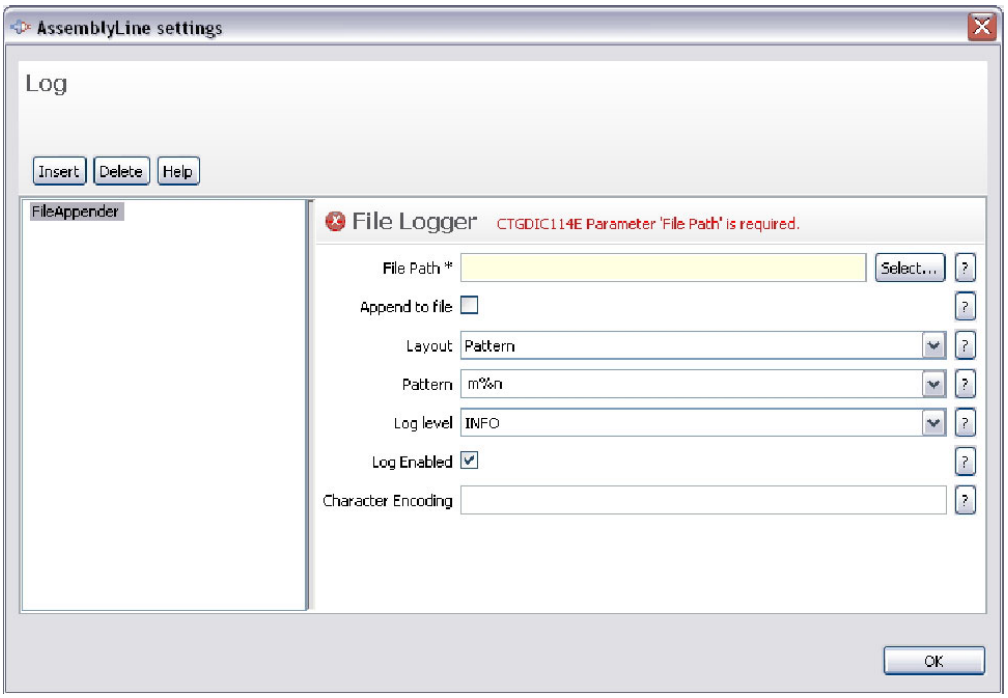


图 16. “AssemblyLine 日志”设置

在该窗口中，可以为该 AssemblyLine 添加记录器。这些记录器不是全局性的，仅针对 AssemblyLine 激活。

AssemblyLine 挂钩

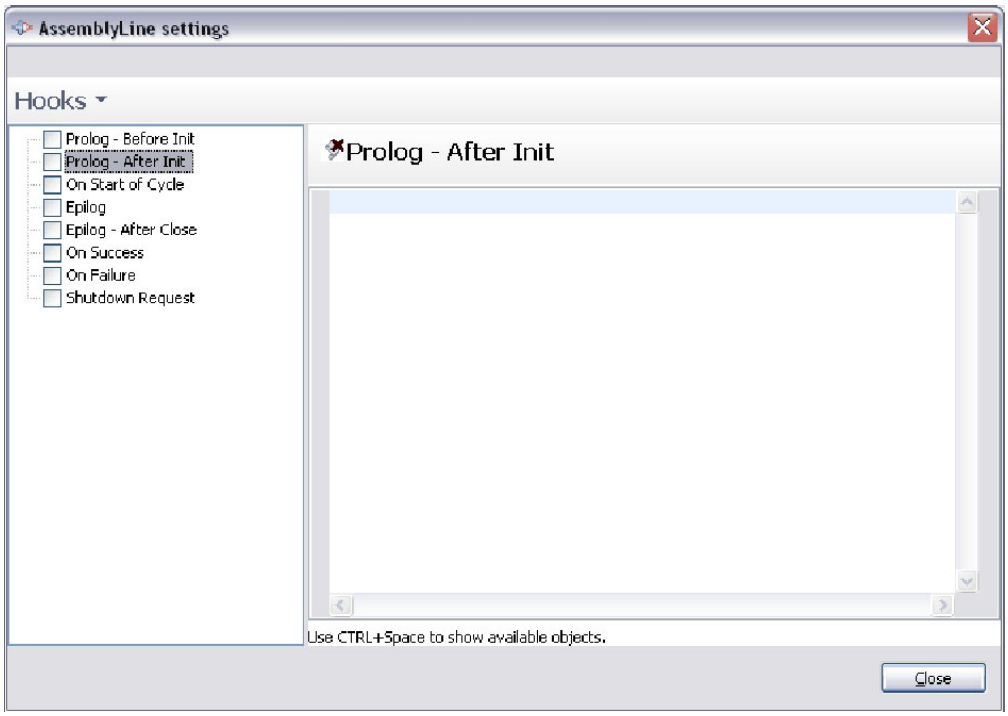


图 17. AssemblyLine 挂钩

在该窗口中，可以启用/禁用AssemblyLine 级别挂钩。已启用的挂钩还将显示在“AssemblyLine”编辑器中的组件面板中。

AssemblyLine 操作

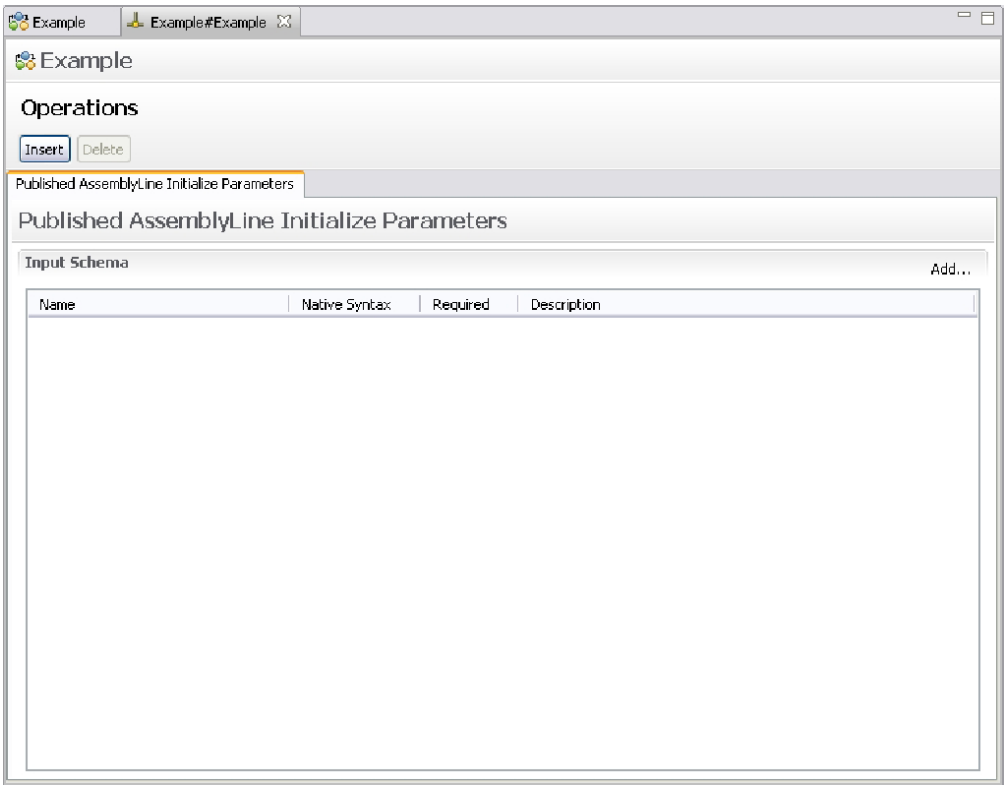


图 18. AssemblyLine 操作

该窗口可使您定义 AssemblyLine 操作。请参阅 参考 中的“Creating new components using Adapters”中的 AL 操作部分，以获取对操作的完整描述。 “插入”按钮可使您添加新操作。**发布的 AssemblyLine 初始化参数**（在缺省情况下可用）是一种特殊操作，用于在初始化组件之前向 AssemblyLine 提供值。

仿真设置

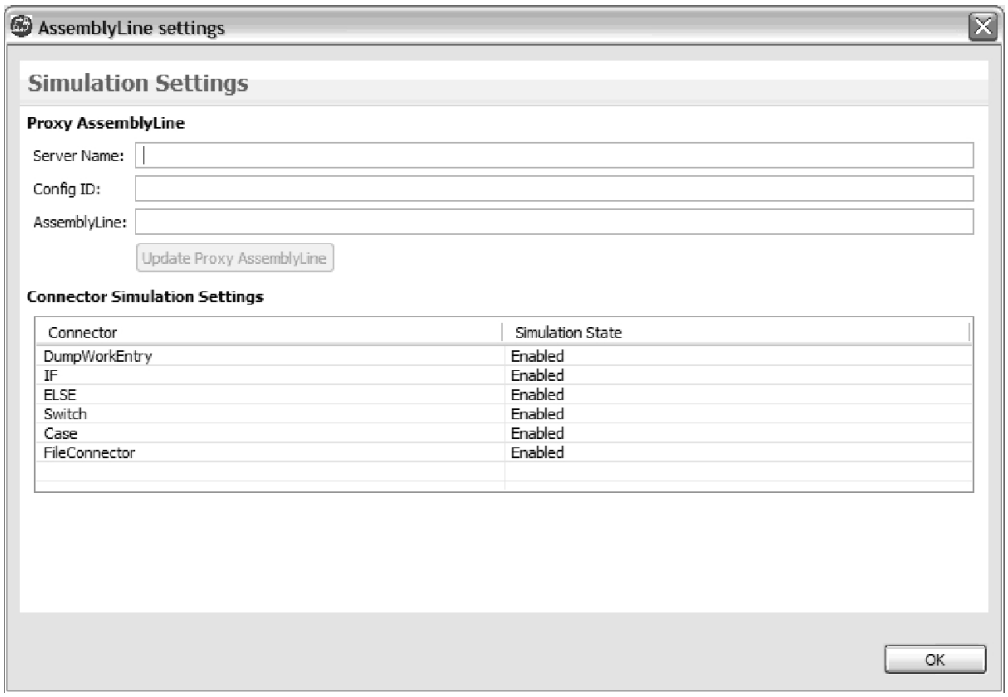
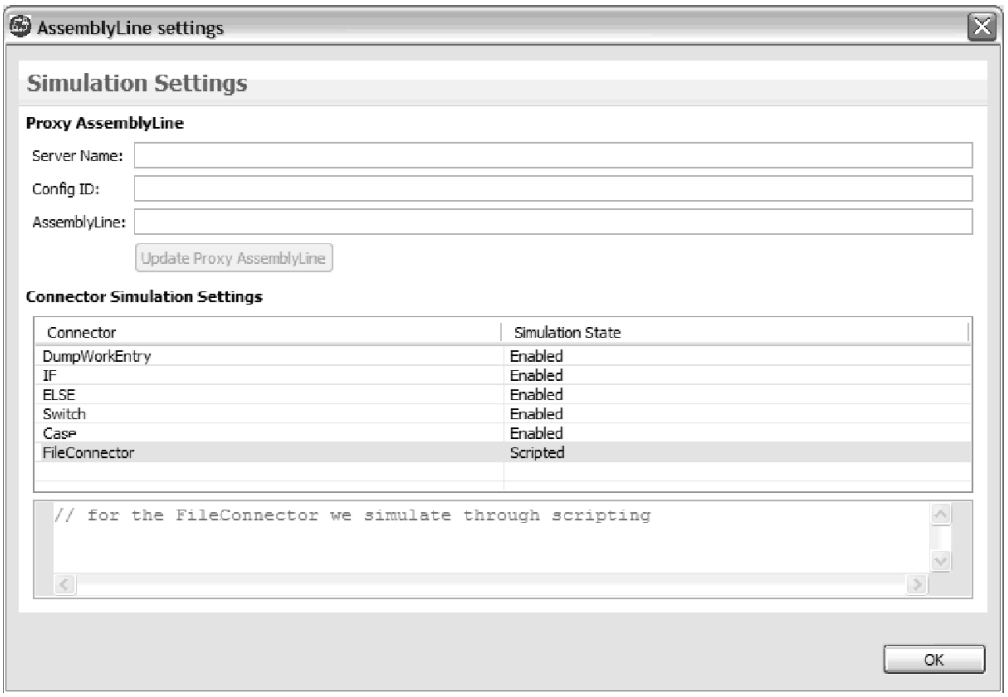


图 19. AssemblyLine 模拟设置

该窗口可使您配置每个组件的模拟设置。关于更多信息，请参阅第 170 页的『AssemblyLine 模拟方式』。当选择对模拟进行脚本编制时，将在该面板底部显示脚本编辑器:



还可以使用**更新代理 AssemblyLine**按钮创建/更新用于模拟代码的代理 AssemblyLine。
图 20. AssemblyLine 模拟设置窗口，具有脚本编辑器

沙箱设置

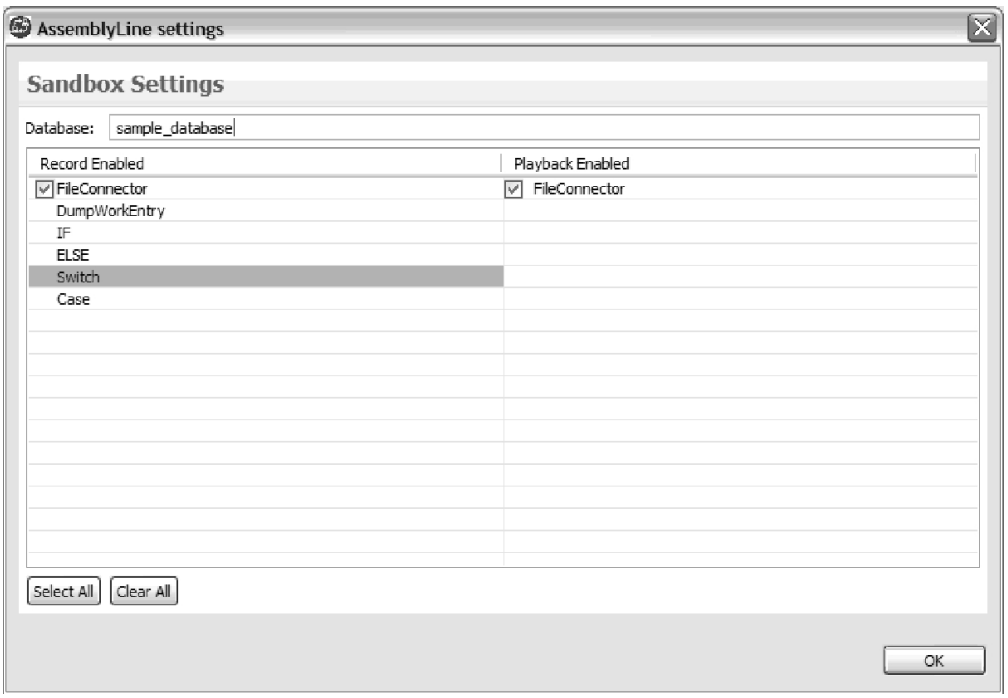


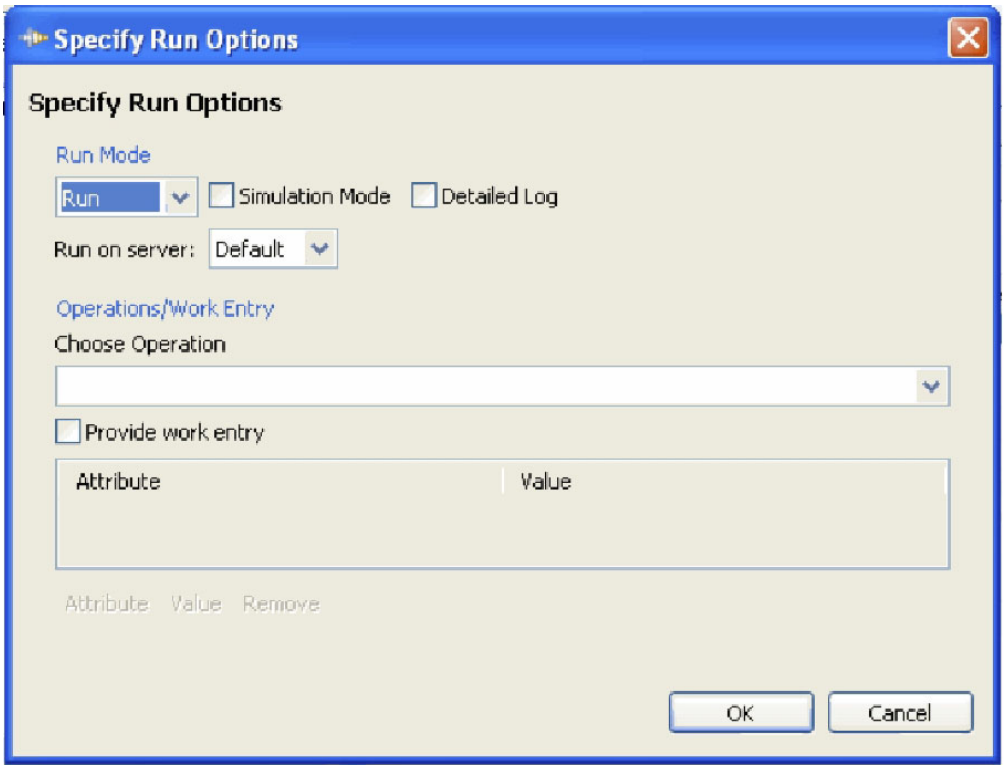
图 21. AssemblyLine 沙箱设置

当您以“记录”或“回放”方式运行 AssemblyLine 时，沙箱设置可使您配置哪些组件将启用记录/回放功能。

有关 IBM Security Directory Integrator 中沙箱功能的更多信息，请参阅第 169 页的『沙箱』。

指定运行选项

通过运行选项对话框，可以配置 AssemblyLine 的运行方式。



如果选择**提供工作条目**，那么可以构建静态条目，此条目会在 AssemblyLine 启动时传入其中。
图 22. "指定运行选项"对话框

组件面板

当打开 AssemblyLine 中的组件时，您将在 AssemblyLine 窗口的下半部分中看到一个快速编辑器面板。

您将看到以下组件的此编辑器:

- 第 90 页的『IF/ELSE/ELSE-IF 分支』
- 第 90 页的『Switch/Case 分支』
- 第 91 页的『For-Each 属性值』
- 第 91 页的『有条件循环』
- 第 92 页的『连接器循环』
- 第 93 页的『属性映射』

IF/ELSE/ELSE-IF 分支

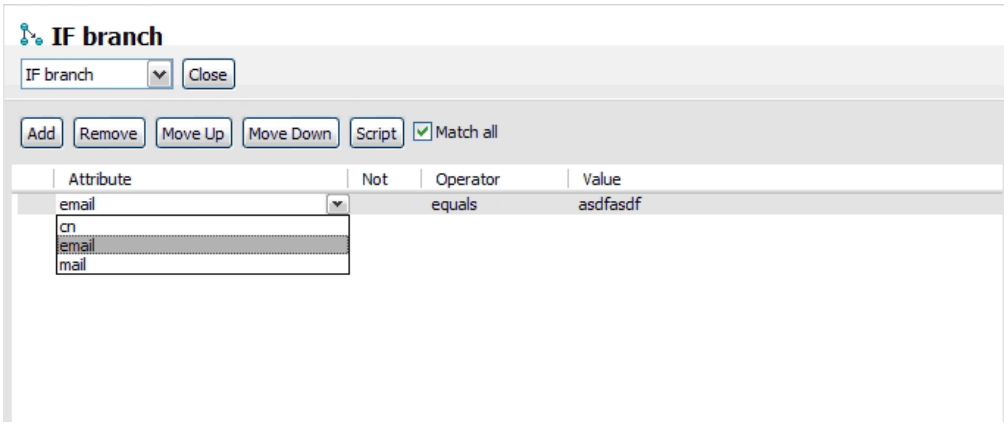


图 23. IF/ELSE/ELSE-IF 分支的快速编辑器

IF 和 ELSE-IF 分支使您能够指定简单表达式和返回 *true* 或 *false* 的定制脚本。使用**匹配全部**复选框在仅当所有条件的求值结果为 *true*（*AND* 方式）时才返回 true。如果未选中，那么仅一个条件必须匹配（*OR* 方式）。

ELSE 分支没有任何参数。

使用**添加**按钮可添加属性/运算符/值控制的新行。您可以使用**删除**按钮除去这些行。值可以为常量或表达式。使用表达式编辑器可通过单击值文本字段后的按钮来配置表达式。移动按钮用于对表达式重新排序。表达式的求值顺序是从顶部到底部。您还可以使用标题下的下拉菜单更改分支类型（即 IF 和 ELSE 等等）。

Switch/Case 分支

Switch 配置有许多选项可供选择值。该值用来与所包含 Case 分支中的值匹配。当这些值相等时将执行 Case 分支。switch 分支会始终执行。

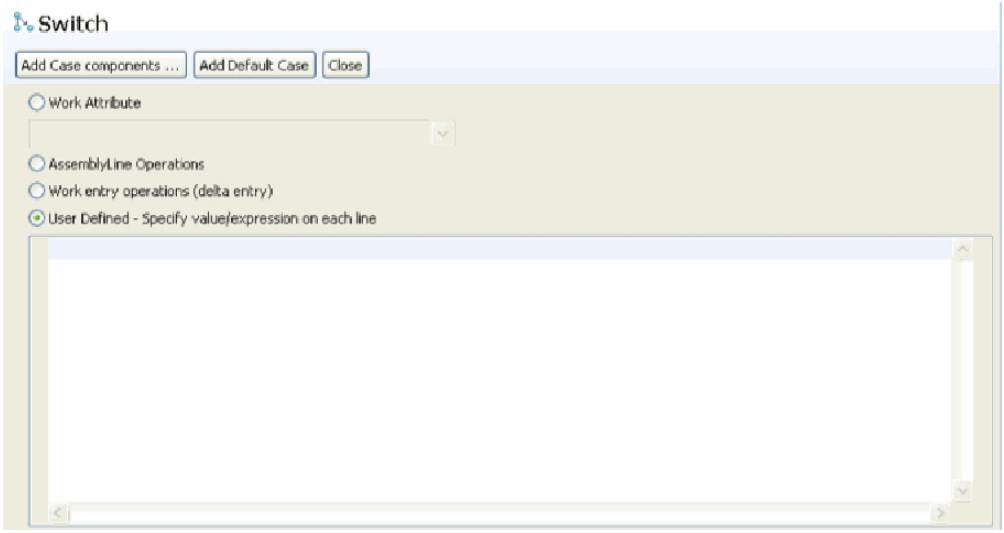


图 24. Switch/Case 分支

可以为 Switch/Case 分支选择的选择选项为:

工作属性
从已知工作属性的列表中选择

AssemblyLine 操作
从已知 AssemblyLine 操作名称的列表中选择。

工作条目操作
使用来自工作条目的操作值（例如，`work.getOperation()` 方法）。

用户定义
您可以在此处指定自己的值。

使用**添加 Case 组件...** 对话框可在此 Switch 分支内生成 Case 分支。根据您的选择，将自动建议您使用那些对选择有意义的值。如果您选择工作条目操作，那么将建议您使用所有已知操作值（例如，添加和修改）。

使用**添加缺省 Case** 按钮可添加缺省 case。如果任何其他 case 分支都不与来自 Switch 组件的值匹配，那么将执行缺省 case。

For-Each 属性值

图 25. 属性值循环

该组件根据属性中的值循环。指定进行循环所使用的工作条目属性名称（工作属性名称）和循环属性名称。循环属性名称设置为来自工作条目属性的当前循环值（例如，`foreach f in work.attr.values; set loopattr = f`）。

有条件循环

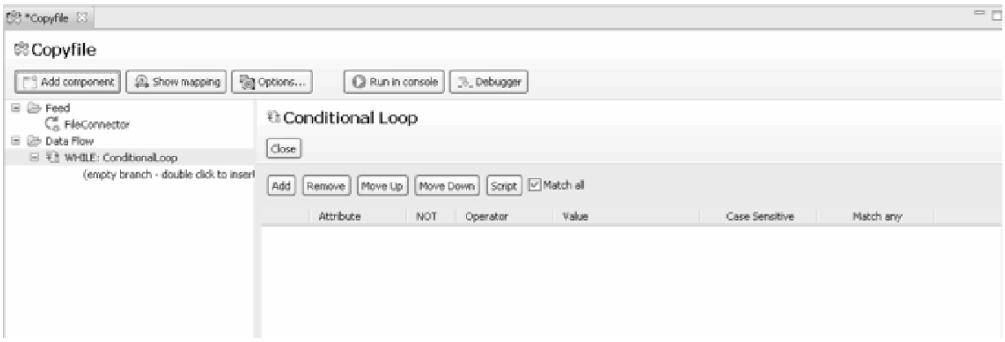


图 26. 有条件循环

有条件循环使您能够指定简单表达式和返回 true/false 的定制脚本。

使用**添加**按钮可添加属性/运算符/值控制的新行。您可以使用“删除”按钮除去这些行。值可以为常量或表达式。使用表达式编辑器可通过单击值文本字段后的按钮来配置表达式。

匹配全部复选框确定是所有的行都必须匹配（选中**匹配全部**）还是仅需要一行为 true 便可执行分支。

连接器循环

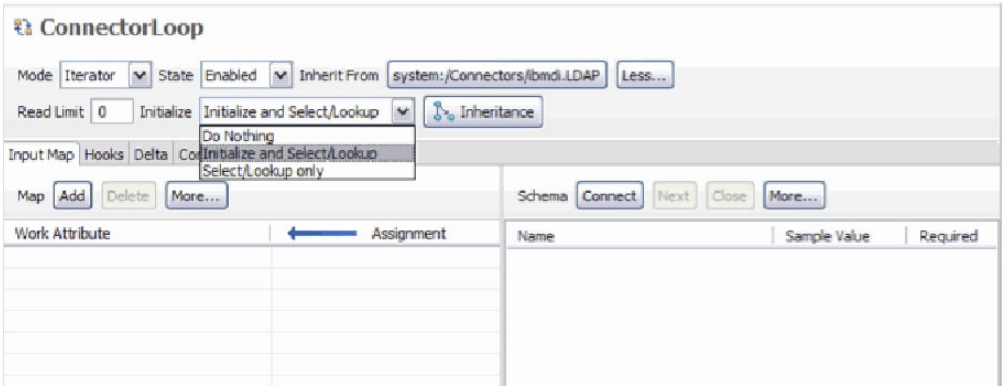


图 27. 连接器循环

连接器循环使用“连接器”编辑器来配置连接器。尽管与上图中的显示类似，但也有一些区别。初始化选项现在显示那些与连接器循环相关的选项而不是标准初始化选项。另外，从“方式”下拉菜单中仅可以选择 Iterator 和 Lookup。

除了看到的连接器的常见选项卡之外，还有一个输出属性映射使您能够配置**连接器参数**选项卡中连接器参数的动态赋值。这与普通输出映射有些不同，因为普通输出映射显示固定模式，即选定连接器的参数的列表。模式部分也没有“连接/下一个”按钮来影响模式。

图 28. 连接器循环中的连接器参数

属性映射

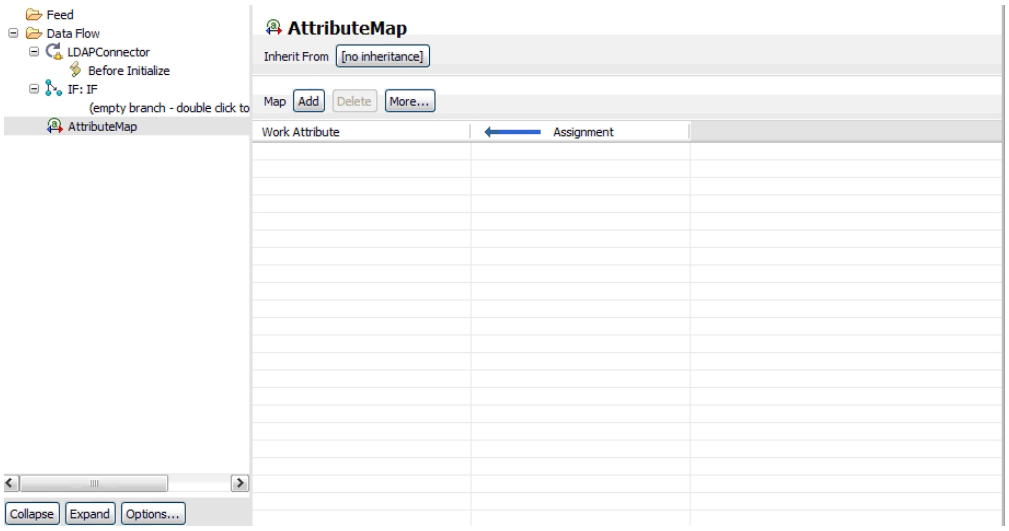


图 29. 独立的属性映射组件

属性映射组件显示可以将属性映射到工作条目的单一面板，与其他组件（例如连接器）内部的属性映射无关。组件还允许复用来自其他组件（例如库中的连接器、函数和其他属性映射组件）的属性映射。有关更多信息，请参阅第 96 页的『属性映射和模式』。

用户文档视图

有时候，AssemblyLine 最终变得非常复杂。为方便他人阅读配置，可以使用文档视图来记录 AssemblyLine 的各部分。

在 AssemblyLine 大纲中，可以右键单击某个组件，然后选择**编辑用户注释**以将文档视图置于最前面:

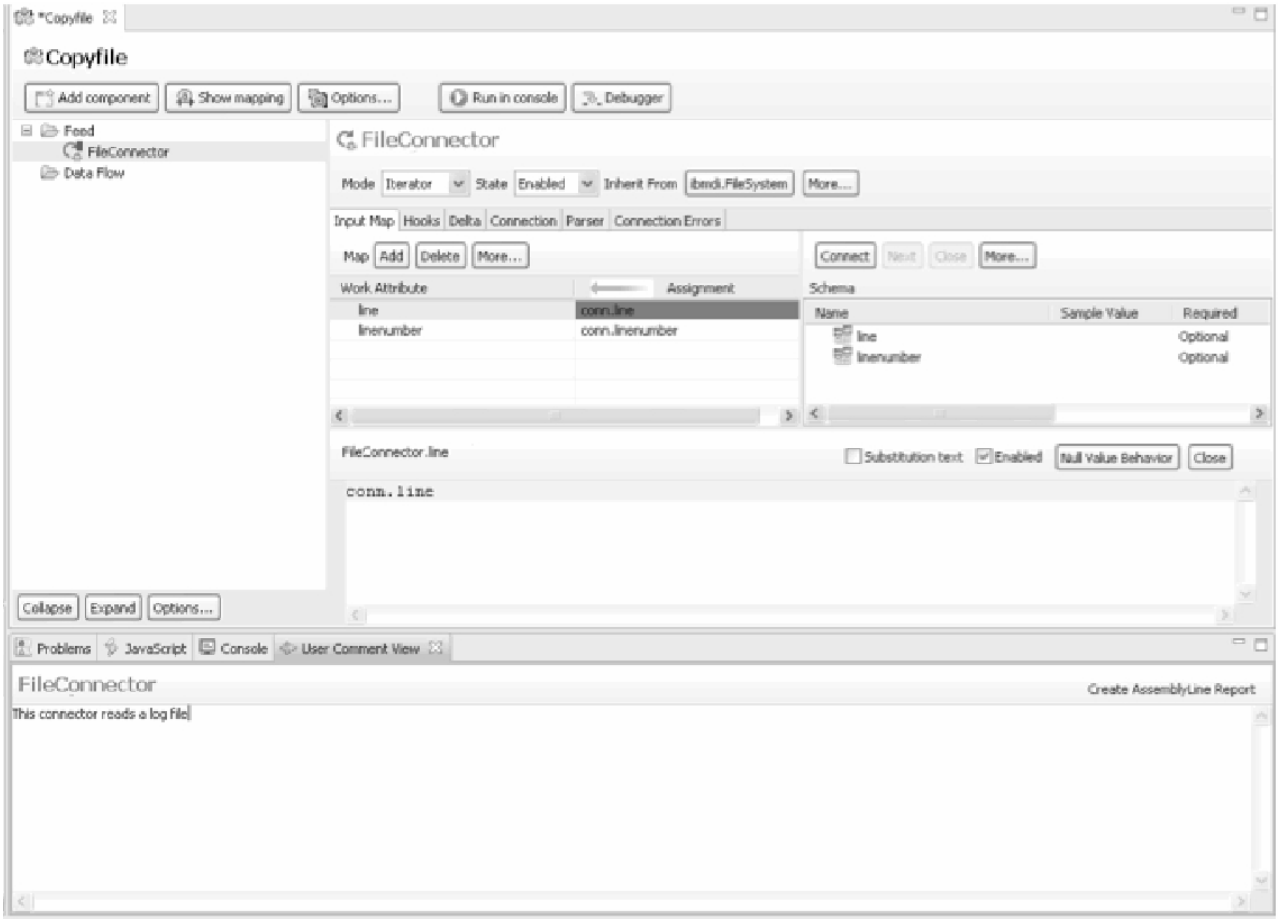


图 30. 用户文档视图

当选择在 AssemblyLine 编辑器中发生变化时，文档视图会反映当前选择。保存 AssemblyLine 时，将会保存您在视图中输入的文本。带注释的组件在其组件图标的左上角进行了装饰

创建 AssemblyLine 报告按钮将创建包含了在 AssemblyLine 中输入的所有用户注释的报告。所使用的报告模板名为 `UserCommentsReport.xsl`，位于 *TDI_Install_dir*`/XSLT/ConfigReports` 目录中。

图 31. 样本 AssemblyLine 报告

运行 AssemblyLine 窗口

运行 AssemblyLine 时，将获得显示了日志输出的窗口。

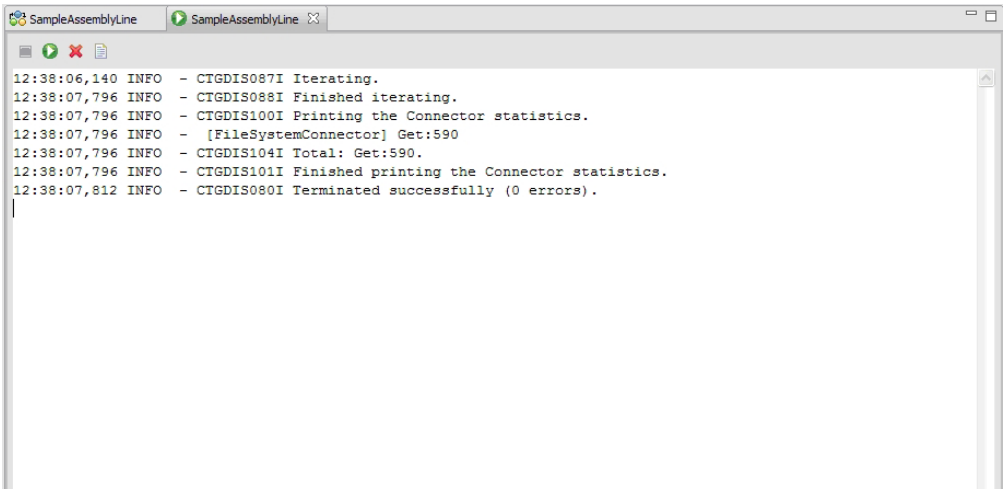


图 32. 控制台日志

在此屏幕中，还可以停止正在运行的 AssemblyLine，并在其终止后重新启动。

注: 停止 AssemblyLine 表示配置编辑器向正在运行 AL 的配置实例（通常是缺省本地服务器）发送了停止通知；该实例不会立即杀死线程，但是一旦服务器重新获得控制权将立即停止执行。这不同于先前的版本，在先前的版本中按下**停止**按钮会导致正在运行 AssemblyLine 的整个服务器进程被杀死。
其余两个按钮用于清除日志窗口，并在单独的编辑器窗口中打开日志文件。该日志窗口仅显示最后几百行以免发生内存不足的问题。

将日志写入前缀为“tdi_ce_al_log”且扩展名为“.log”的临时文件。该文件位于特定于平台的临时目录中，该目录通常由 TEMP/TMP 环境变量定义。关闭“运行 AssemblyLine”窗口时该日志文件将自动删除，但是如果应用程序或机器崩溃，可能需要手动除去这些日志文件。用于该文件的编辑器缺省设置为简单文本编辑器，但是可以通过将“.log”扩展名映射至其他编辑器（包括外部编辑器）对该编辑器进行更改。使用 **Windows > 首选项**菜单选项可打开下列对话框:

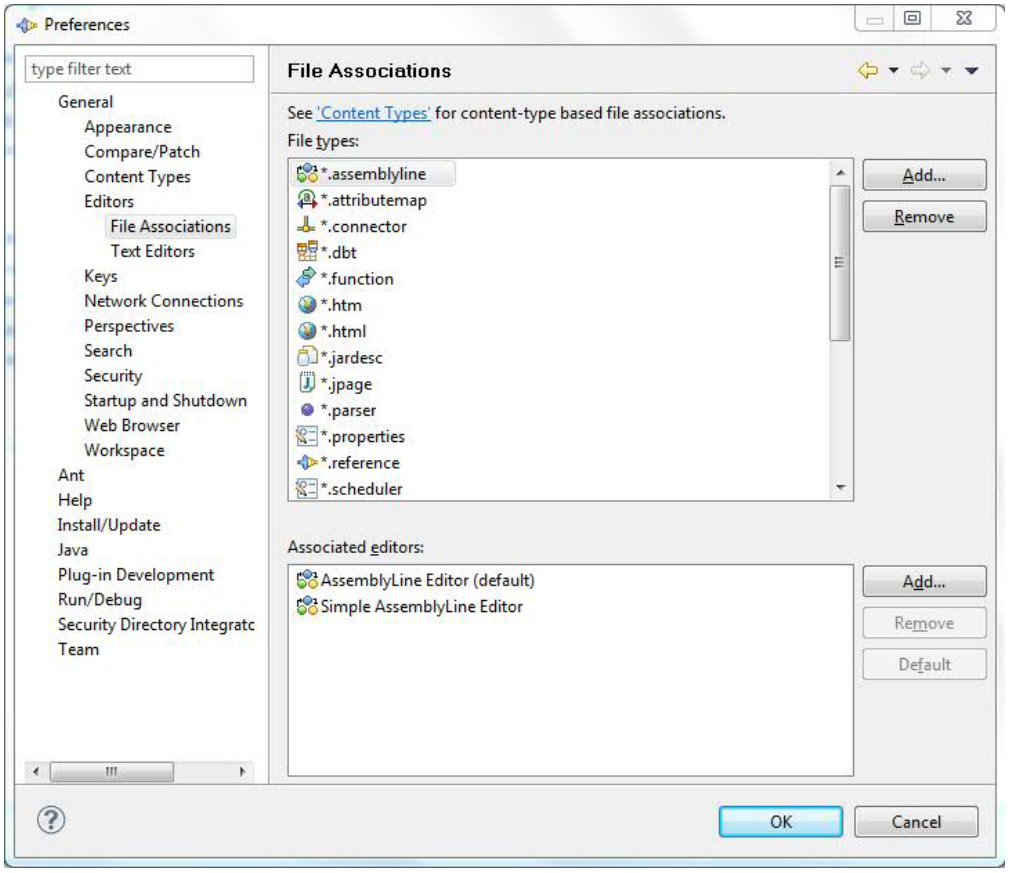


图 33. “配置编辑器文件”关联首选项

您可以在此处添加“.log”扩展名并将其与编辑器相关联。

属性映射和模式

属性映射是使用 AssemblyLine 或组件编辑器中的属性映射面板来完成的。

在 AssemblyLine 编辑器中，可以添加属性，方法是右键单击属性映射部分并选择添加属性，或者使用如下所示的工具栏中的**添加**按钮。

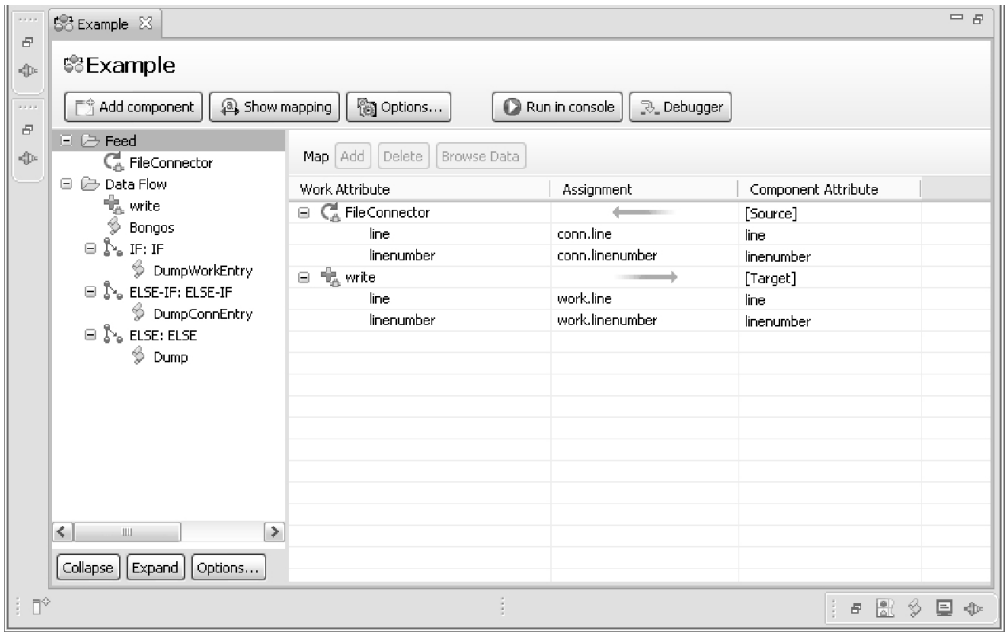


图 34. 属性映射

在该窗口中，看不见 AssemblyLine 中组件的模式。要使用该模式，请通过在左侧树中选择该组件来打开该组件的编辑器。

属性映射的常见场景是首先发现组件的模式。当发现模式时，CE 将运行后台作业以执行组件的查询模式方法。如果未返回任何模式，CE 将询问您是否想要读取一个条目以尝试从该条目中派生模式。然后将结果回填至正在编辑的组件的模式。

下图显示了发现属性后组件的输入模式内容。如果组件由于某种原因未向您提供模式，那么可以使用工具栏上的**添加...** 按钮手动添加模式项，或使用**更改继承**选项重用其他组件配置的模式。

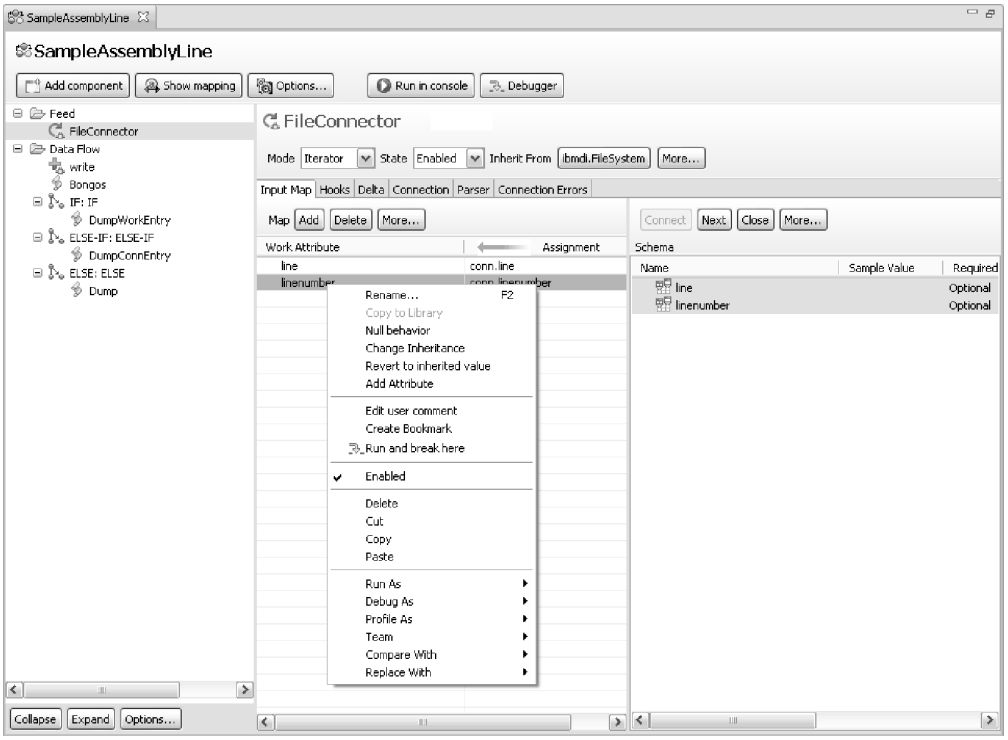


图 35. 具有已发现属性的属性映射

还可以使用标题栏上的下拉菜单更改模式配置的继承。

使用模式，可以将单个项拖放至属性映射，或使用上下文菜单中的**映射属性**功能并根据需要修改映射。

图 36. 更改属性映射继承

注: 拖放功能取决于窗口化环境中的特定范围。尤其对于 UNIX 系统，公共桌面环境（CDE）不提供该功能，因此为了设置映射，将需要使用上下文菜单中的**映射属性**功能，

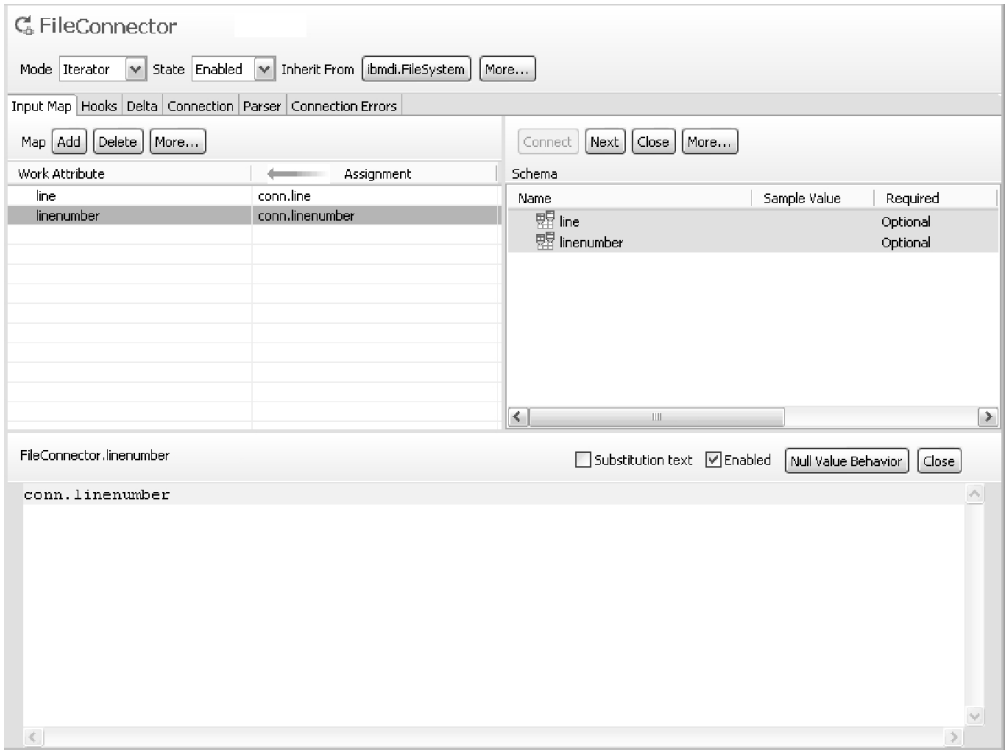


图 37. 属性映射（具有针对单独属性的 JavaScript 编辑窗口）

如果没有任何模式或者想要添加独立于该模式的属性，那么您完全可以这样作。使用**添加**按钮以将新属性添加至映射。为该属性命名，并将“conn.*attribute-name*”或“work.*attribute-name*”的表达式指定给该新属性。可以在“AssemblyLine”编辑器和“连接器”编辑器窗口中执行该操作。

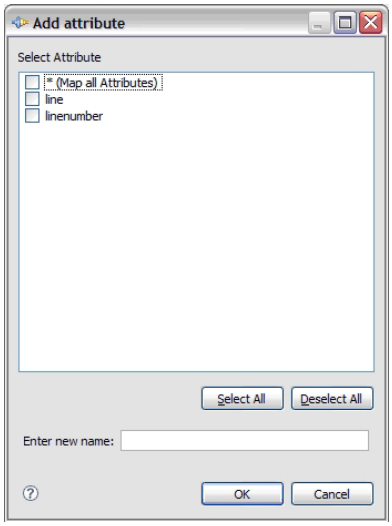


图 38. “添加属性”对话框

对话框显示可编辑文本字段，其中可以输入该新属性的名称。以上列表包含模式中所有已知属性名；您可以选择想要添加至属性映射的属性名。

当将更多组件添加至 AssemblyLine 时，可以在组件之间拖动属性以使其更有意义。将一个组件拖至另一个组件上会使所有已映射属性映射至目标组件。还可以将属性从属性映射中拖至左侧面板中的组件上，该面板中显示了 AssemblyLine 中的所有组件。这样将执行所拖拽的所有这些项的一个简单映射。这类似于将属性拖至属性映射面板中的组件上。

在《入门》中，“属性映射”的概念是相当广泛的，且具有很多示例。

根据连接器及配置它的方式，在“连接器”配置窗口中将会有不同的选项卡。

- 处在支持从已连接系统进行输入的方式下的连接器将具有名为**输入属性**的部分。
- 处在支持输出到已连接系统的方式下的连接器将具有名为**输出属性**的部分。
- 部分连接器支持可进行输入和输出的方式。如果按该方式进行配置，那么您将看到**输入属性**部分以及**输出属性**部分。

外部属性映射

属性映射可以从外部属性映射文件进行继承。外部属性映射文件是包含属性映射项的文本文件，就如同您在实际映射屏幕中所具有的文件。区别在于外部文件使用不同于内部 XML 结构的格式。这使您能够更轻松地为任何连接器配置属性映射，甚至不必进入 CE。CE 在属性映射的继承对话框中提供以下选项:

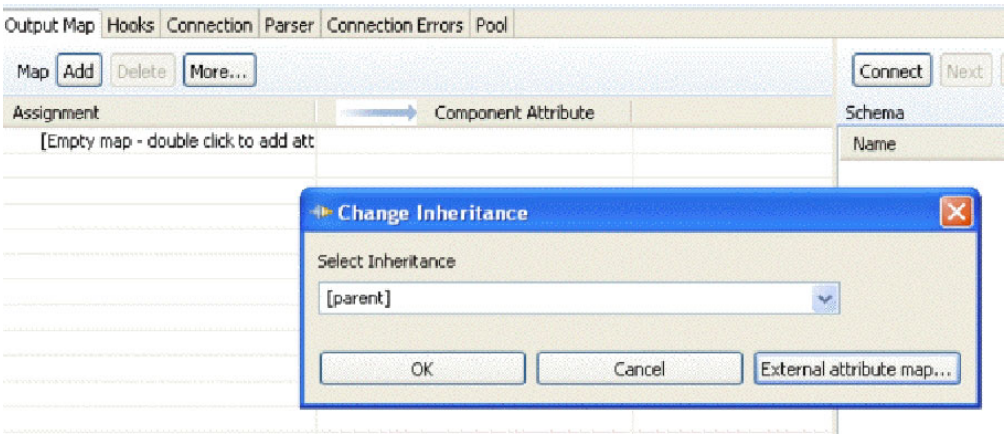


图 39. 属性映射: 继承对话框

单击**外部属性映射...**按钮以选择现有文件，或者输入“`file:`”，后跟属性映射文件的完整路径。如果要使用相对路径名，请以点加斜杠 (*./*) 作为文件名前缀。

输入属性映射:

输入属性映射是实现从输入源到 AssemblyLine 中工作条目的数据移动的过程。输入属性映射显示在连接器的“属性映射”窗口中，其出现在“连接器编辑器”中时带有一个从称为“[源]”的实体指向连接器的箭头。其也显示在输入属性映射下的“模式”窗口中。

开始之前

为了可以设置“输入属性”映射，必须在连接器的工具栏中将连接器设置为可支持输入的方式。可支持输入的方式一般为 Iterator、Lookup 和 Server。

然后，在**输入映射**部分中，您从输入源中选择希望在 AssemblyLine 中处理的那些属性。

关于此任务

要为“输入属性”映射设置的连接器可以驻留在 <workspace>/Resources/Connectors 中，或者驻留在 AssemblyLine 内其指定的位置。

过程

1. 单击**输入映射**。
2. 单击**连接**，再单击**下一步**以获取众多数据源的模式。某些连接器或“连接器 – 解析器”组合具有预定义的模式，而其他的会提示您从数据源读取示例条目并进行检查以发现属性。
3. 最后，从**模式**列表中选择属性，然后将其拖入“属性映射”中，或者通过“添加”和“删除”按钮手动添加这些属性。“属性映射”控制哪些属性会引入到 AL 中进行处理，以及所指定的任何变换。

下一步做什么

从数据源检索这些已映射的属性，并将其置于*工作*条目中，然后传递到 AssemblyLine 中流程部分的后续连接器中。

如果没有在 AssemblyLine 中直接创建连接器，那么为了在 AssemblyLine 中使用此连接器，请将连接器从其在 <workspace>/Resources/Connectors 中的位置拖动到 AssemblyLine 的**供给**部分。

输出属性映射:

输出属性映射是完成数据移动的过程，它将数据从 AssemblyLine 中的工作条目移动到已连接系统中的输出目标。在连接器编辑器中启动输出属性映射后，它将显示在连接器的“属性映射”窗口中，并带有一个箭头从连接器指向称为“[目标]”的实体。它们还显示在输出属性映射下的“模式”窗口中。

开始之前

为了可以设置输出属性映射，连接器必须设置为支持输出（连接器的工具栏）的方式。输出的典型方式是 AddOnly.。一些方式（像 CallReply）支持输入和输出。

然后，在“输出属性”部分的**输出映射**中，从 AssemblyLine 的工作条目选择想要输出到已连接系统的那些属性。

关于此任务

为输出属性映射设置的连接器可以驻留在 <workspace>/Resources/Connectors 中，或驻留在 AssemblyLine 中的指定位置。但是，如果连接器仅在 <workspace>/Resources/Connectors 中（即不是 AssemblyLine 的成员），那么您无法轻易地将工作条目属性拖动到输出属性映射中。在此情况下，请将连接器拖入到 AssemblyLine 中，或者手动创建映射，方法是单击“属性映射”窗口中的**添加**按钮，或者也可以右键单击“属性映射”窗口中的连接器，然后选择**添加属性映射项**。

过程

1. 单击**输出映射**。
2. 单击**连接**以获取数据源的模式。一些连接器或连接器解析器组合有预定义的模式，此时将显示这些模式。然而，许多连接器没有预定义的模式。

3. 如果连接器在 AssemblyLine 中，请将先前映射的工作条目属性拖动到 AssemblyLine 编辑器的“属性映射”窗口中的连接器上。或者，手动创建属性 – 将在运行时执行名称匹配。例如，以 `some_attribute` 为名称创建的输出映射属性映射项将导致名为 some_attribute 的工作条目属性映射到名称相同的已连接系统属性。

下一步做什么

这些映射的属性是在 AssemblyLine 的流程中调用该连接器时从 *Work* 条目中检索到的，并输出到已连接的系统。

如果您未直接在 AssemblyLine 中创建连接器，那么为了在 AssemblyLine 中使用该连接器，请将连接器从 `<workspace>/Resources/Connectors` 中其所在的位置拖动到 AssemblyLine 的**流程**部分。

连接器编辑器

当您在 AssemblyLine 的连接器中编辑连接器文件或使用**编辑**功能时，会使用连接器编辑器。

在 『创建连接器』 部分中对创建连接器进行了概述。

编辑器使用您在连接器的向导和弹出对话框中找到的相同窗口小部件。编辑器由 6 个选项卡组成，在这些选项卡中显示连接器各个方面的配置面板。顶部是连接器的主要属性，例如它的方式、状态和其他常规连接器选项。

这些选项卡为:

1. 第 103 页的『输入和输出属性映射』
2. 第 104 页的『挂钩』
3. 第 105 页的『连接』
4. 第 105 页的『解析器』
5. 第 106 页的『链接条件』
6. 第 107 页的『连接错误』
7. 第 109 页的『变化量』
8. 第 110 页的『池』
9. 第 111 页的『连接器继承』

创建连接器

创建连接器包含决定连接器所驻留的位置以及指定给该连接器的初始参数。

开始之前

决定连接器应该驻留的位置；此位置处于以下两个位置中的某个位置:

1. 用于重用和资源共享的连接器驻留在 `<workspace>/Resources/Connectors` 目录下。此处通常是创建和维护连接器的最佳位置。通过将定义该方法的连接器拖至相应位置可将这些连接器添加至 AssemblyLine。

   之后，可以改变连接器的这些设置，以便其在 AssemblyLine中扮演其指定的角色，但是将保留主要的继承设置，以便不更改“资源”部分中这些设置的定义。

2. 还可以直接在 AssemblyLine中创建连接器；定义该方法的连接器是仅在该特定AssemblyLine的上下文中有效的特别定义。

关于此任务

连接器形成了使用 IBM Security Directory Integrator 创建的任何解决方案的主干，它们建立与想要与之交换数据的系统的连接。

过程

1. 在工作空间中单击右键并选择**资源 > 连接器 > 新建连接器...**，或选择**文件 > 新建 > 连接器**
2. 导航至想要新连接器所在的位置并对该新连接器命名。
   a. 推荐的位置是 `<workspace>/Resources/Connectors`。
   b. 或者，可以直接在 AssemblyLine中创建新连接器。浏览至目标 AssemblyLine 中的位置：Iterator 和 Server 方式连接器的“馈入”部分或者其他所有方式的流。
3. 单击**完成**以创建连接器。
4. 在**连接**选项卡中，将新连接器的方式设置为预期方式。
5. 在**连接**选项卡中设置该连接器的连接参数；必需参数使用星号（*）标记。有些连接器还需要在**解析器**选项卡中配置解析器。
6. 转至连接器配置窗口中的**属性映射**窗口以发现或定义该数据源的模式：单击**发现属性**以获得该数据源的模式。某些连接器或连接器解析器组合具有预定义模式。在这些模式下，不会提示您从数据源中读取样本条目，并检查该数据源以发现属性。

下一步做什么

一旦定义连接器，将可以设置称为*属性*的信息的哪些部分将流入 AssemblyLine，或从 AssemblyLine 中流出。从 AssemblyLine 的角度来说，此过程称为*属性映射*。因此，通过输入连接器从已连接系统到 AssemblyLine 中工作条目的映射属性定义将在*输入属性映射*中执行；相反，通过输出连接器从工作条目到已连接系统的映射属性将在*输出属性映射*中执行。

输入和输出属性映射

“属性映射”选项卡显示组件的输入和输出属性映射以及模式。

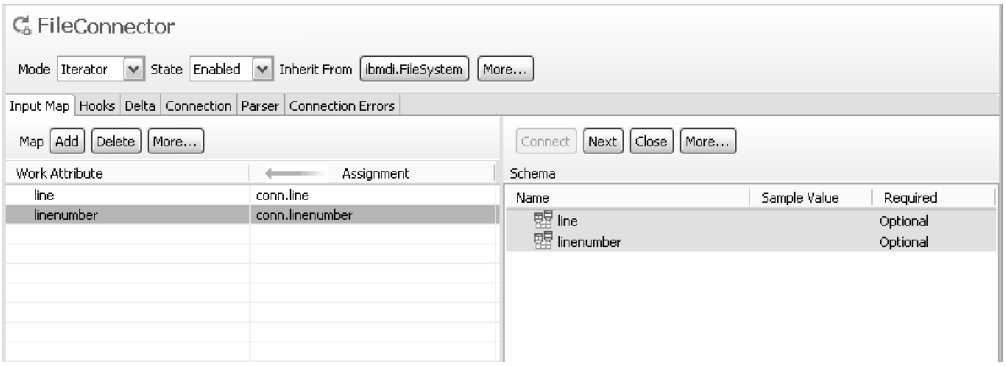


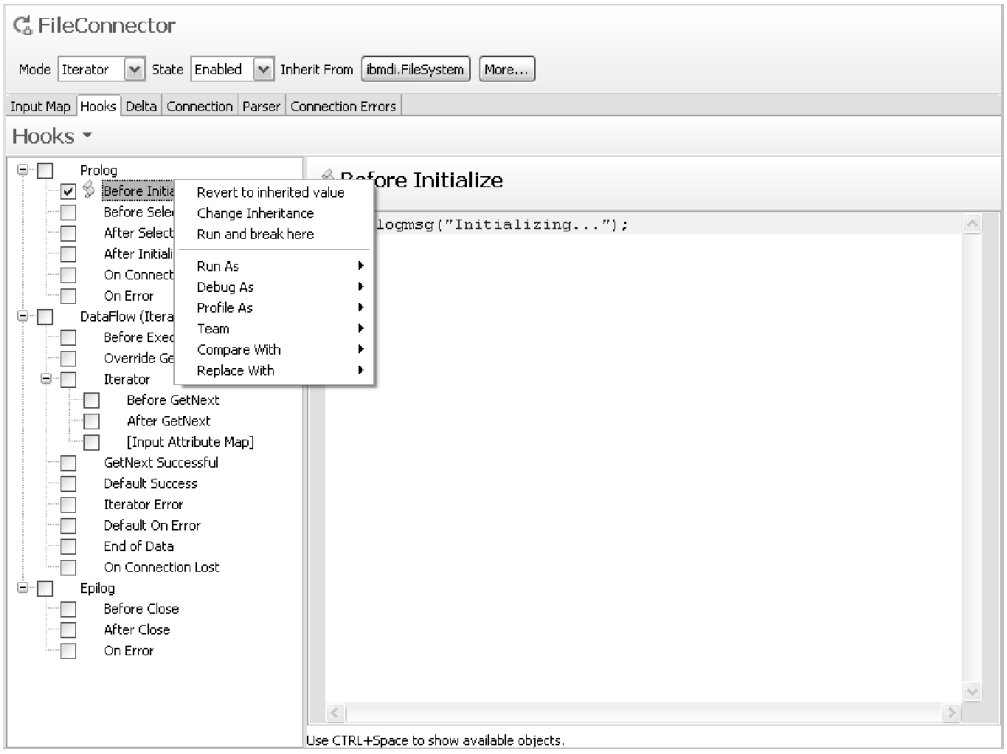
图 40. “属性映射”窗口

请参阅“AssemblyLine”编辑器中的第 96 页的『属性映射和模式』这一部分，以获取该窗口的描述。

挂钩

“挂钩”选项卡显示该连接器的所有挂钩。

使用该复选框可启用/禁用挂钩，并选择该挂钩以编辑其内容。当您修改挂钩的内容时，将自动启用该挂钩。包含脚本代码的挂钩在树形视图中具有脚本图标，以便能够快速标识挂钩是否有内容。请注意，如果启用了挂钩，那么当挂钩到达执行流时将执行该挂钩，而不论其是否包含任何脚本。



请参阅第 27 页的『AssemblyLine 流程和挂钩』这一部分，以获取对于 AssemblyLine 级别以及单个组件级别的各种挂钩的讨论。

连接

图 41. “连接”选项卡

“连接”选项卡中的参数通常特定于想要配置的组件。请参阅 *参考* 中组件的各个规范。

解析器

如果连接器可以使用（或需要）解析器，您还将具有该解析器的选项卡。

使用**选择解析器**工具栏按钮可更改连接器的解析器。

例如，行阅读器解析器具有如下所示的配置屏幕:

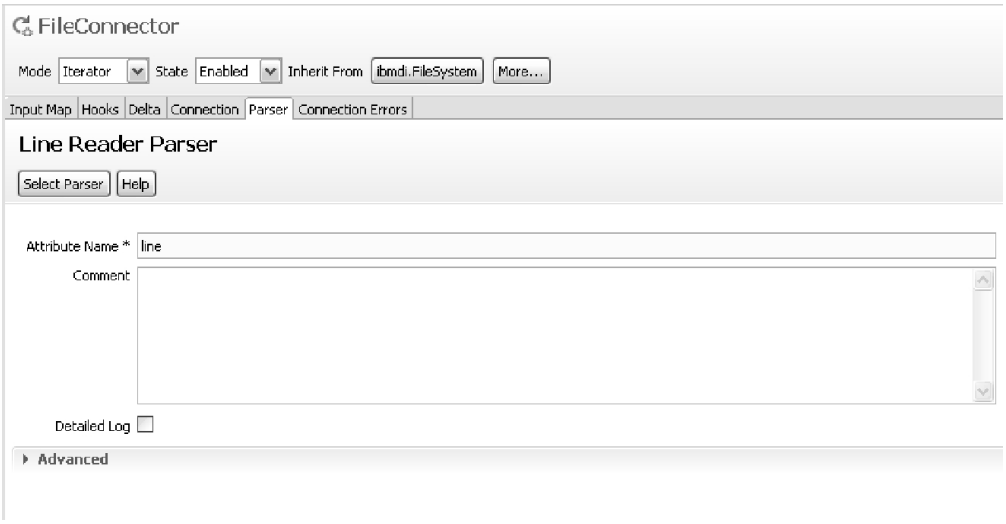


图 42. 行阅读器解析器

链接条件

组件需要链接条件时，您将看到“链接条件”选项卡。

组件实际是否展示“链接条件”选项卡，不仅取决于组件类型，还取决于其方式。关于更多信息，请参阅第 17 页的『链接条件』。

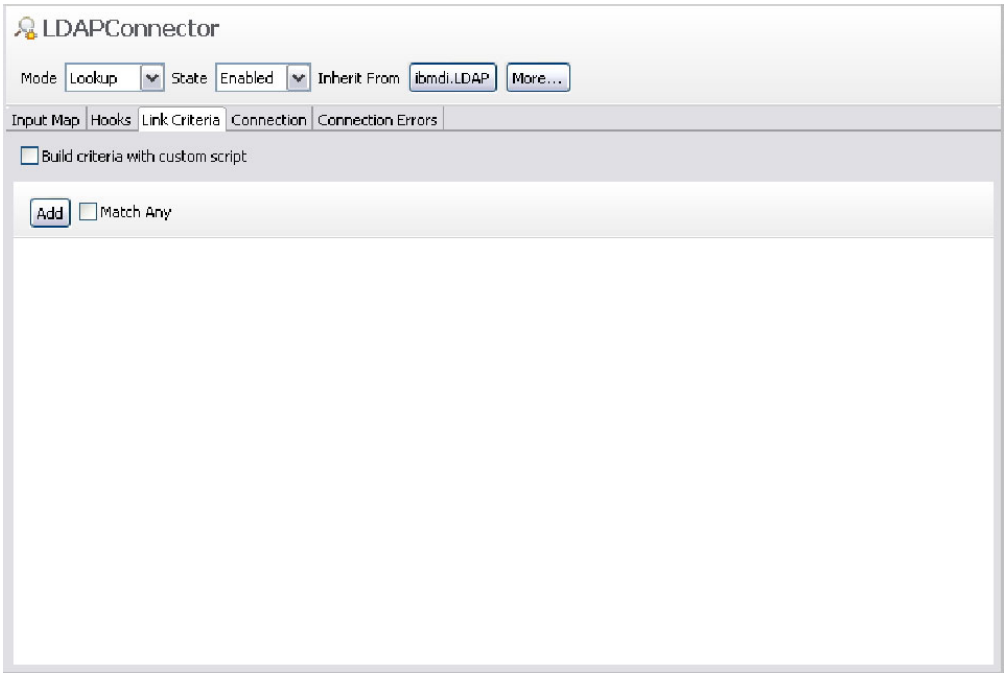


图 43. “链接条件”选项卡

使用**添加**按钮来将新的控件行添加到视图。使用**删除**按钮来除去单个行。在视图中，您指定属性名称、操作数（例如: 等于、包含）和匹配值。匹配值可以为常量或表达式。使用  按钮来启用表达式编辑器:

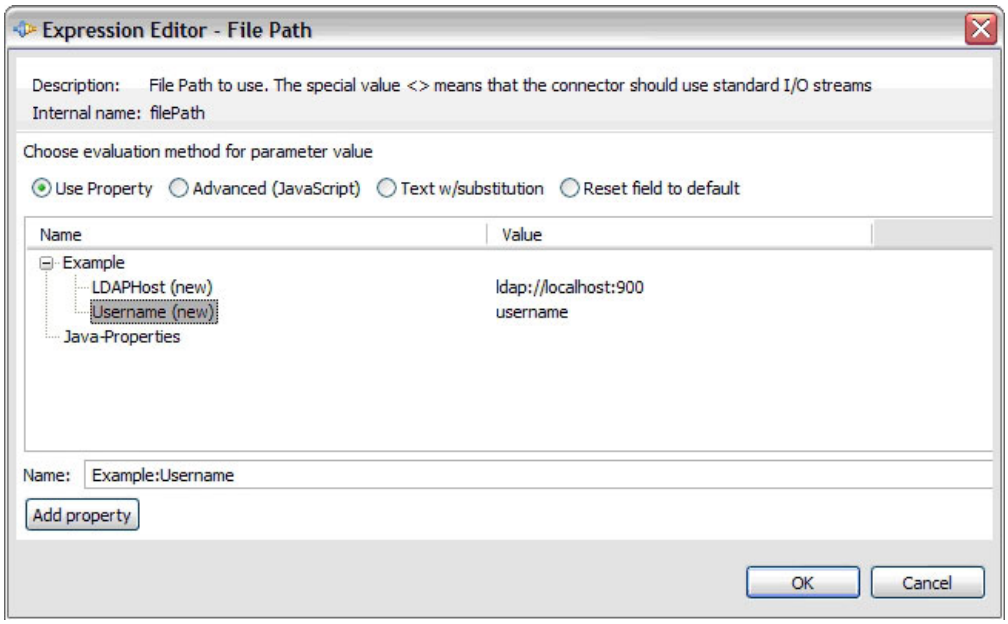


图 44. “表达式编辑器”窗口，简易方式

在表达式编辑器中，可以方便地从某个属性文件选择属性，或者使用高级方式，在此方式中您返回基于 JavaScript 代码的值。选中**高级 (JavaScript)** 复选框以在属性选项和 JavaScript 代码之间切换。您只能使用两者之一。

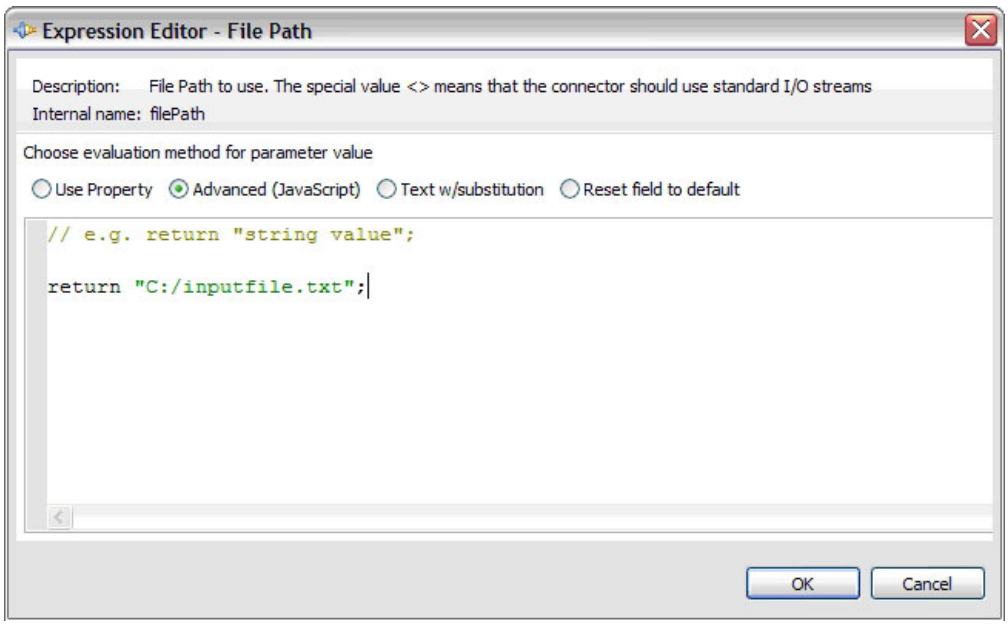
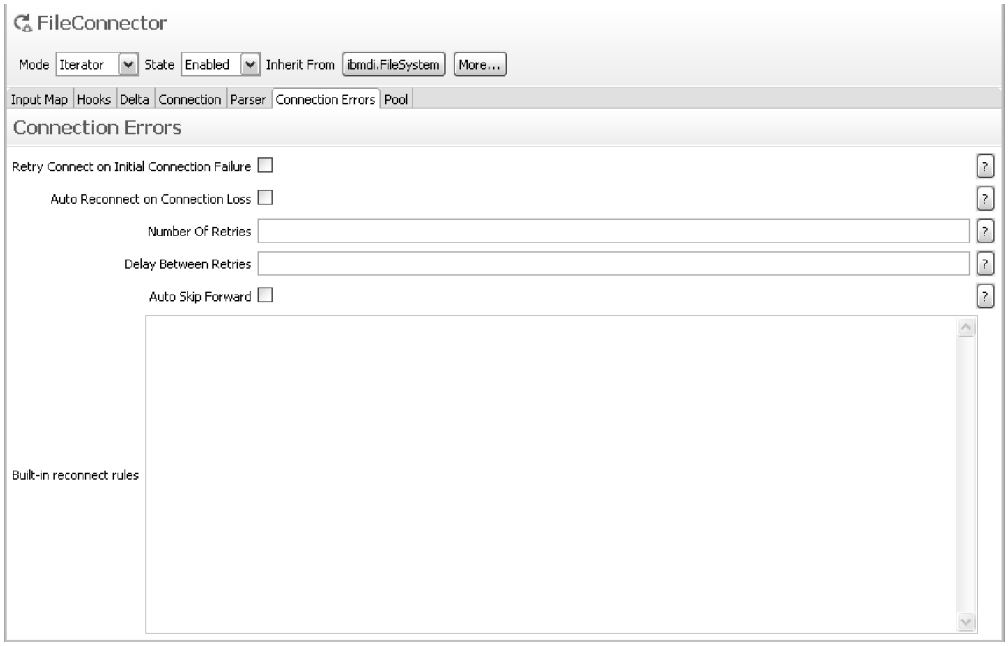


图 45. “表达式编辑器”窗口，高级 (JavaScript) 方式

连接错误

“连接错误”选项卡是配置连接灵活性的位置，配置连接灵活性是当组件建立的连接失败时发生的一种自动化行为。

或者，当您选择**在初始连接失败时重试连接**时，可以配置连接尝试是否将在启动时失败的情况下抛出异常，且是否将重试。如果设置了此标志，且在初始化连接器时无法建立连接，那么将进行“重新连接”尝试；不是真正地重新连接，因为最初未建立连接，但是一般来说，当建立的连接丢失时可以发生与情境相同的机制。



对于已建立的连接，其余的参数具有以下意义:

连接丢失后自动重新连接

如果设置了该标志，且在初始化连接器后连接丢失，那么将进行重新连接尝试。

重试次数

发生问题时，在放弃之前将进行重新连接尝试的次数。如果稍后发生了新的问题，将进行相同次数的尝试。

两次重试之间的延迟

在首个重新连接尝试之前，每个重新连接尝试之间所等待的秒数。

自动跳转至

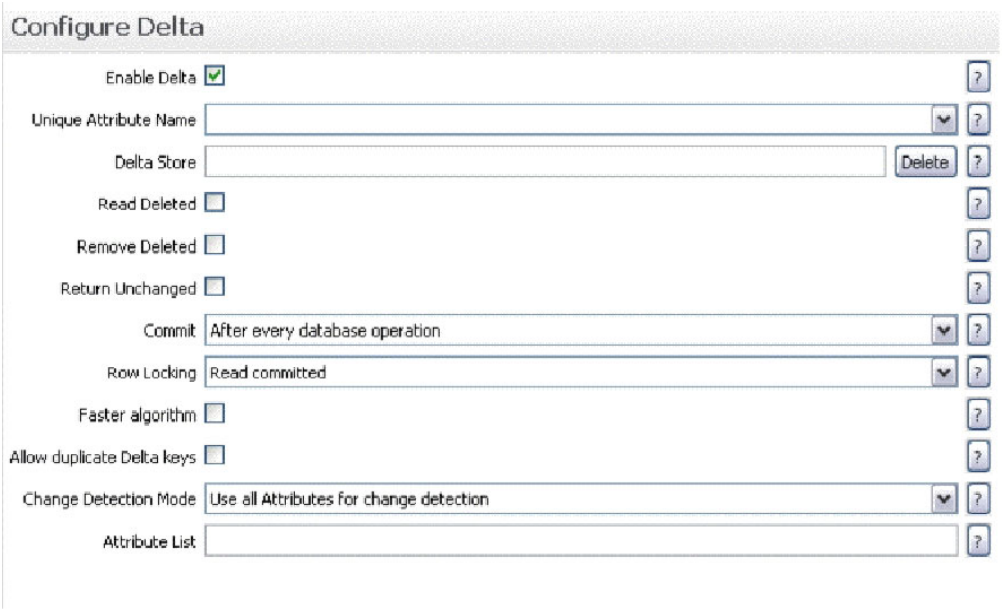
重新连接后，自动跳转与成功读取次数相同的次数。

内置重新连接规则

这些规则与“重新连接规则引擎”相关联；请参阅*安装与管理*中的相应部分

，以获取更多信息。

变化量



该选项卡仅适用于 Iterator 方式。

该选项卡中的参数具有以下意义:

启用变化量

这是此连接器的变化量引擎的主开关。如果未选中该复选框，将不启用以下参数。

唯一属性名称

此为某个属性或多个输入属性（以“+”分隔）中所选项的名称，在给定数据源中具有唯一值。具有重复密钥的数据源无法受变化量函数的控制，除非启用了**允许重复的变化量密钥**。关于更多信息，请参阅第 13 页的『Delta 检测』。

变化量存储

系统存储中的表保存自以前运行此连接器以来的变化量信息，以便能够检测后续运行中的差异。

读取已删除的条目

如果选中该复选框，AssemblyLine 会将已删除的条目注入迭代器完成迭代（即，完成输入）时运行的 AssemblyLine。操作码将指示此条目已从输入源中删除。请注意，除非同时启用了“除去已删除项”标志，否则不从变化量存储中除去标记为删除的条目。

除去已删除的条目

如果选中该复选框，从输入源中删除的条目也将从变化量存储中删除，从而在后续运行中不会再次检测到这些条目。

返回未更改的条目

如果选中该复选框，该运行中任何未经更改的条目都将注入 AssemblyLine。

落实 选择何时落实因通过输入进行迭代而对变化量存储所作的更改。选项有:

- 在每次数据库操作后
- 在 AL 循环结束时

- 在连接器关闭时
- 不自动落实

缺省值为**每次数据库操作以后**。

行锁定 选择针对变化量存储器连接的事务隔离级别。此参数通过设置*事务隔离级别*来处理多个 DB 客户机访问相同数据时变化量存储器表中的行锁定需要。设置更高的隔离级别可以通过使用行锁定和表锁定来减少事务异常（称为“脏读”、“不可重复读”和“幻读”）

选项有:

- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

缺省值为 READ_COMMITTED。有关更多信息，请参阅第 194 页的『行锁定』部分。

属性列表

此为以逗号分隔的属性的列表，在计算更改过程中将检测或忽略这些属性的更改。所列属性中的更改将受到**更改检测方式**参数的影响，此参数指定是要忽略还是检测这些更改。有关此参数和下一个参数的更多信息，请参阅第 195 页的『仅检测或忽略特定属性中的更改』部分。

更改检测方式

指定要检测还是忽略**属性列表**参数中所列属性的更改。可能的值是:

- IGNORE_ATTRIBUTES
- DETECT_ATTRIBUTES
- DETECT_ALL

更快的算法

选中该复选框时，指示 AssemblyLine 通过消耗更多内存使用更快的算法来计算更改。

允许重复的变化量密钥

如果在长期运行的组装流水线中对 Changelog/Change Detection 连接器启用了变化量功能，那么可以多次修改条目。这些修改将导致重复接收条目，致使抛出重复变化量键异常。通过选中此参数，具有重复键属性（在**唯一属性名称**参数中指定）的条目可由已启用变化量的 Iterator 连接器来处理。

池

仅当连接器驻留在连接器库中时连接器的池定义（即，Project/Resources/Connectors 中的文件）才可见。从 AssemblyLine 中打开的连接器将不具有该选项卡。

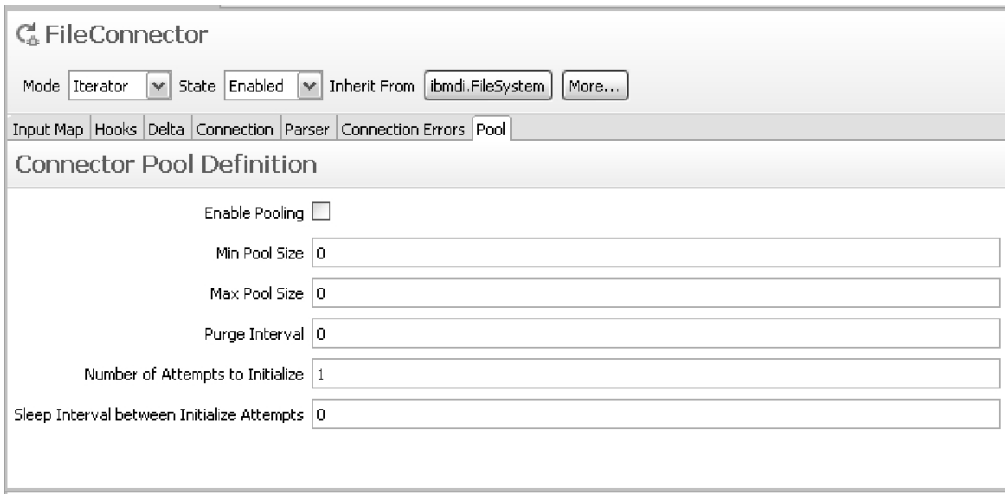


图 46. “池”选项卡: “连接器池”定义

合用的连接器用于 AssemblyLine 时，将显示以下选项卡:

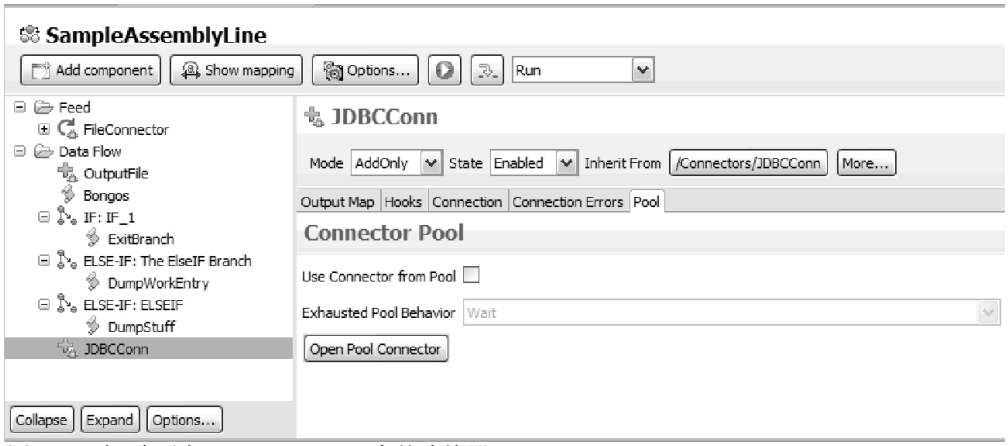


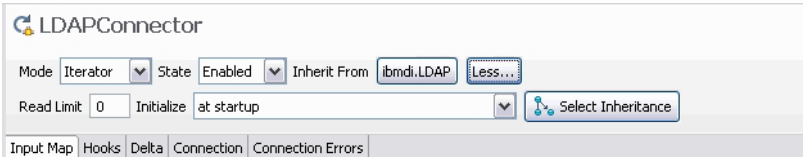
图 47. “池”选项卡: AssemblyLine 中的连接器

使用**打开池连接器**按钮可打开合用的连接器。

连接器继承

您可以在编辑器中的各个位置更改继承，还可以使用连接器继承按钮以获取继承设置的完整列表。

使用**更多...**按钮以展开连接器编辑器的标题，然后单击**继承**按钮。



继承按钮将启动显示连接器的所有继承设置的对话框:

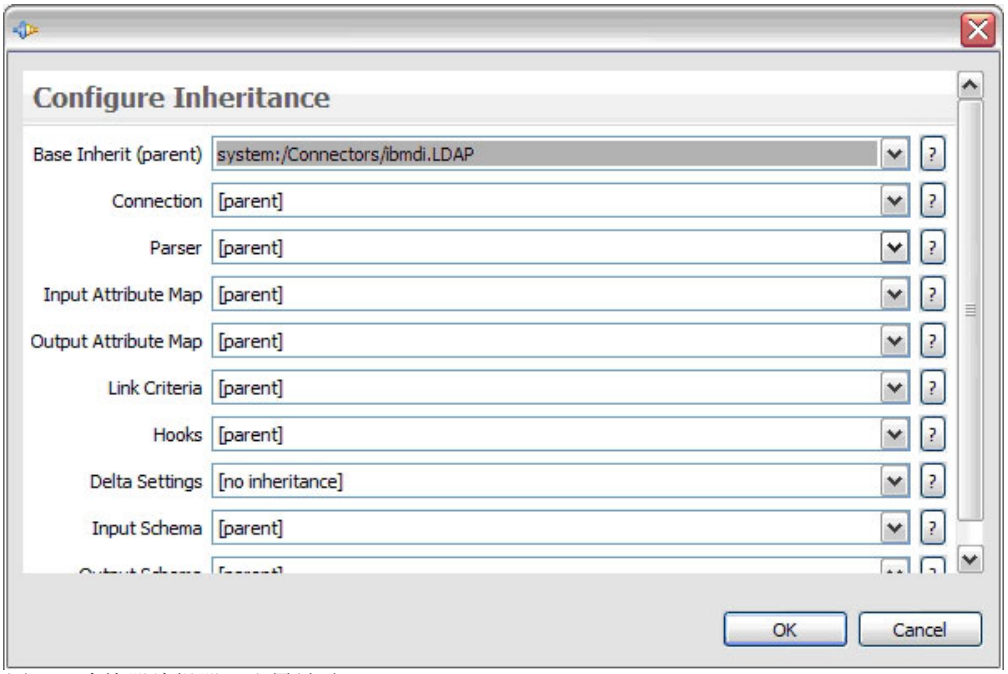


图 48. 连接器编辑器：配置继承

表示法 [parent] 表示从**基本继承（父代）**字段中列出的组件继承的项。

服务器编辑器

服务器编辑器用于定义如何连接 IBM Security Directory Integrator 服务器。

一台服务器总是由 CE 进行定义并指定为“缺省”。在 IBM Security Directory Integrator 服务器视图中，可以向项目中添加新服务器。

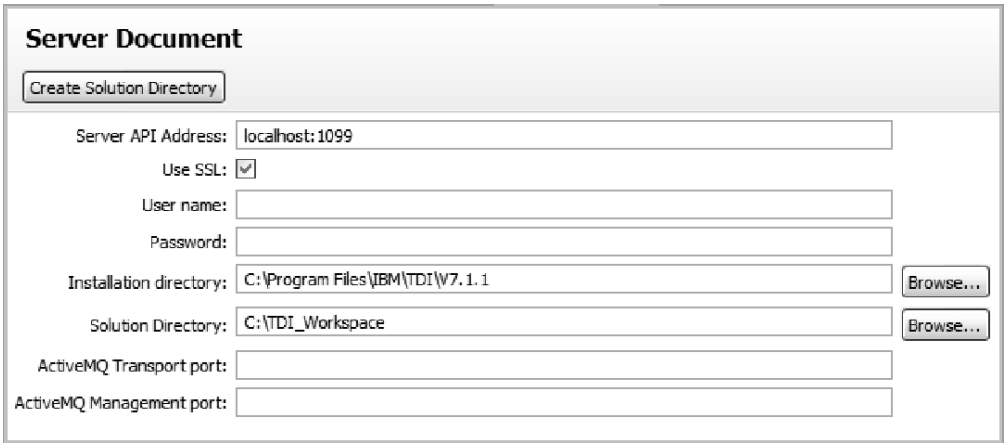


图 49. 服务器文档编辑器

服务器 API 地址是指 IBM Security Directory Integrator 服务器的主机:端口地址。如果指定安装目录，那么 CE 可以启动服务器。启动新 IBM Security Directory Integrator 服务器之前，需要配置所有参数并使用**创建解决方案目录**按钮为此服务器创建必要的文件。还可以指定用于 Apache ActiveMQ 传输和管理的特殊端口。

模式编辑器

模式编辑器用于管理设计模式文件。

这些文件可供其他编辑器中的输入和输出映射使用。模式为仅设计时（即仅在 CE 中适用），并且通常在您具有大量不希望在运行时配置文件中显示的模式的情况下使用。

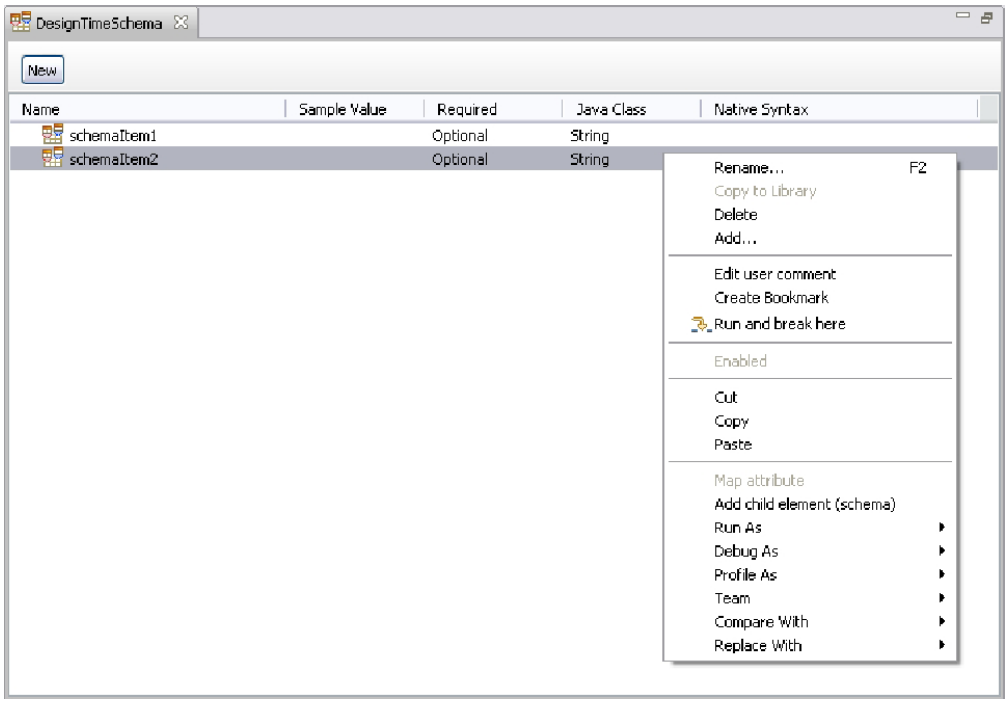


图 50. 模式编辑器

编辑器提供**新建**按钮来添加新顶级模式项和上下文菜单，以对现有模式项进行操作。

数据浏览器

数据浏览器可以更深入查看目标系统。当前只有 LDAP 和 JDBC 连接器提供连接器的额外详细信息。通过右键单击库或 AssemblyLine 中的连接器可打开数据浏览器。

在导航器中，可以单击右键并选择**浏览数据**以打开新的编辑器窗口，其中可以在连接器设置的当前连接中浏览数据。

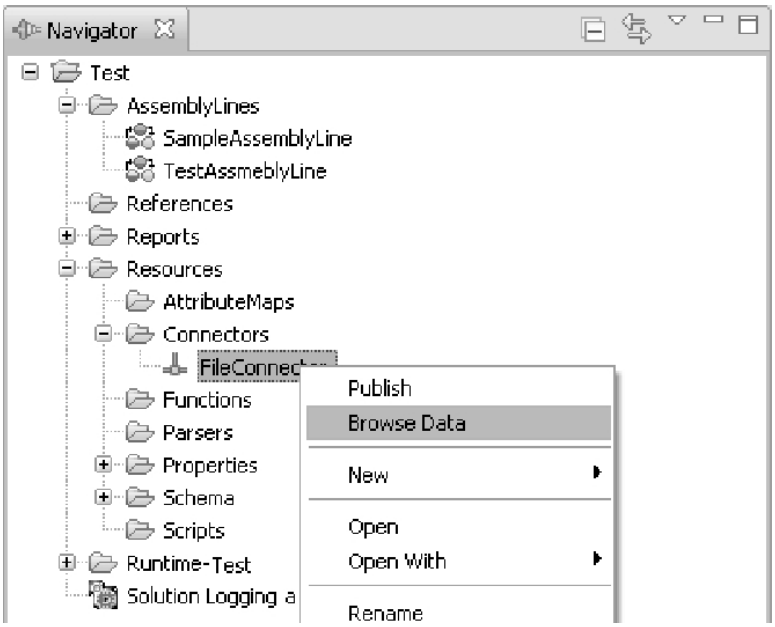


图 51. 数据浏览器

在 AssemblyLine 中，可以执行相同操作，并为数据浏览器打开一个新窗口:

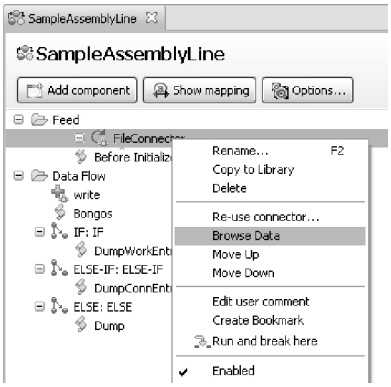


图 52. 数据浏览器

一般数据浏览器

这是用于配置编辑器对其没有明确了解的那些连接器的数据浏览器。其提供一种简单方式来从数据源浏览结果集。

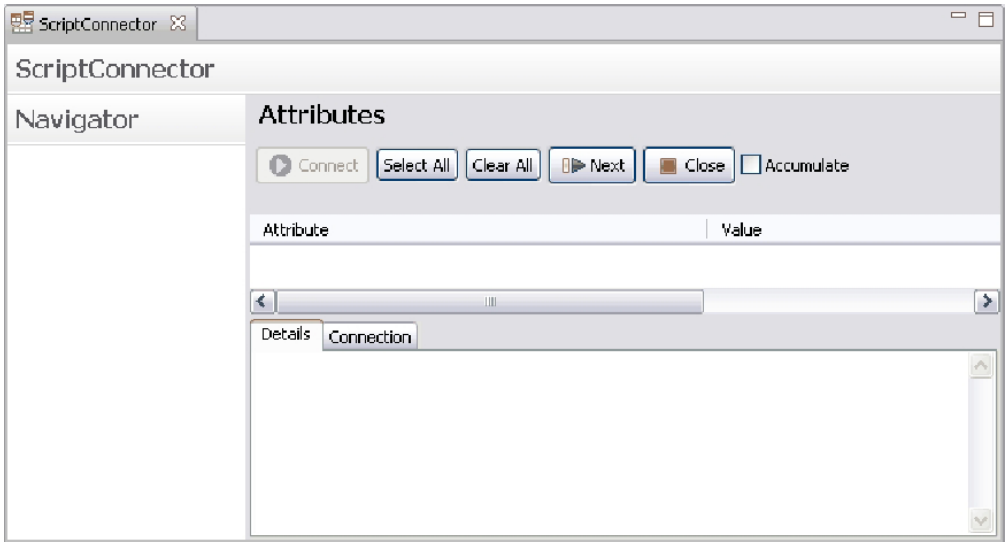


图 53. 一般数据浏览器

上面部分显示从连接器读取的属性列表以及工具栏。列表中的属性有复选框；选中某个属性时，将使用简单的 `conn.attributename -> work.attributename` 表达式来为此属性创建属性映射。取消选中某属性时，将除去此属性的属性映射。

工具栏有以下功能:

表 8. 数据浏览器工具栏

全部切换	此命令将切换列表中每个属性的复选框。这将导致对所列的全部属性的属性映射进行修改。
累积	此命令切换是否累积所发现属性的列表。累积属性时，从连接器读取的每个记录将与现有属性列表合并。不累积时，将除去所有属性，然后才在列表中显示下一记录。
	单击此按钮以关闭连接。您关闭浏览器的“编辑器”窗口时，将自动关闭连接。如果您已在此编辑器中修改了连接设置，那么在读取下一记录之前，您一般要关闭此连接。
	单击此按钮以从连接器读取下一记录。没有从连接器返回的更多记录时，在工具栏左侧会显示消息以指示: 已没有来自连接器的更多条目。在此情况出现后，再按此按钮会使连接器开始从其结果集的开头进行读取。

屏幕的下半部分显示两个选项卡。第一个选项卡是**详细信息**选项卡，其包含有关当前选择的详细信息。对于一般数据浏览器，此选项卡始终为空。

第二个选项卡是**连接**选项卡。此选项卡显示连接器的连接配置。您可以修改连接参数并将其保存，就如同您在打开“连接器”编辑器时所执行的操作一样。

流数据浏览器

当连接器使用解析器时，将使用基于流的数据浏览器。基于流的数据浏览器将首先初始化连接器，然后尝试获取输入流并在详细信息选项卡中显示输入数据中前 20K 的数据。

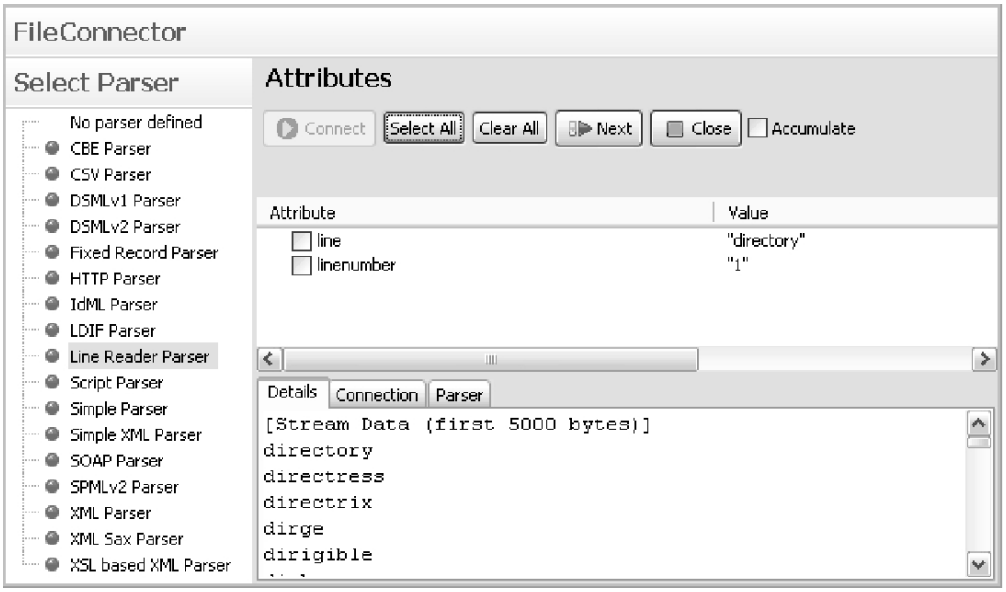


图 54. 流数据浏览器

左边现在包含可以选择解析器的**选择解析器**列表，以便尝试读取输入流的内容以查看其是否与您的期望匹配。当选择解析器时，将使用该解析器的配置表单更新**解析器**选项卡。无论何时更改解析器，都将关闭连接器以便您可以容易地查看解析器是否可以通过连续选择读取下一个前的解析器来解释输入。

注: 当选择表中的解析器时，还可以修改连接器配置以使用该解析器。当关闭和保存（或者使用**文件 > 保存**功能）时，将用这些当前已配置的参数有效地更新配置。

JDBC 数据浏览器

JDBC 数据浏览器在左侧显示所有表和视图。“详细信息”选项卡显示此列表中选定项的信息。

图 55. JDBC 数据浏览器

详细信息选项卡显示从 JDBC 连接对象获取的系统信息。左侧的树形视图还显示所有表和视图及其列（作为子条目）。选择表或视图时，将以整个表的语法填充“详细信息”选项卡。例如，如果选择“IDI_PS_DEFAULT”表，那么应看到类似如下的内容:

图 56. JDBC 表详细信息

选择表或视图中某列时，您将仅查看此列的详细信息。

图 57. “用作表”选项

还可以右键单击表名称并使用**用作表**功能来更新 JDBC 连接器配置的 **Table Name** 参数。

JDBC 表头的工具栏中的  按钮执行对连接的重新发现。如果初始发现已失败，或者如果您已在“连接”选项卡中更改 JDBC URL，那么您只需使用此按钮。

LDAP 数据浏览器

LDAP 数据浏览器显示 LDAP 服务器提供的模式和上下文前缀（搜索条件）。

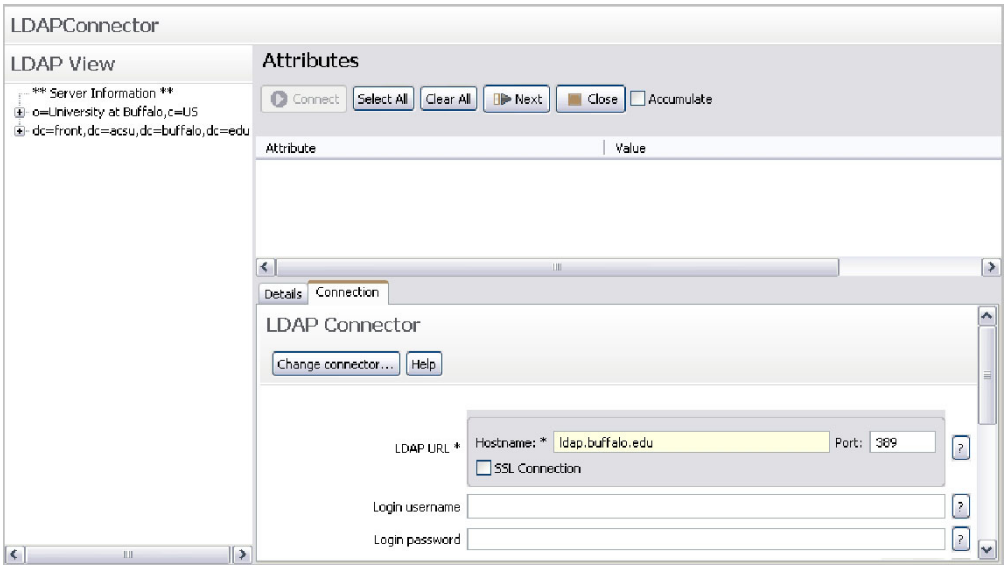


图 58. LDAP 数据浏览器

LDAP 视图包含 LDAP 服务器提供的服务器信息和搜索条件。根据您在此树中的选择，您将看到不同的结果。如果选择某个非模式节点，那么您应该在“详细信息”选项卡中看到此特定条目的详细转储，如下所示:

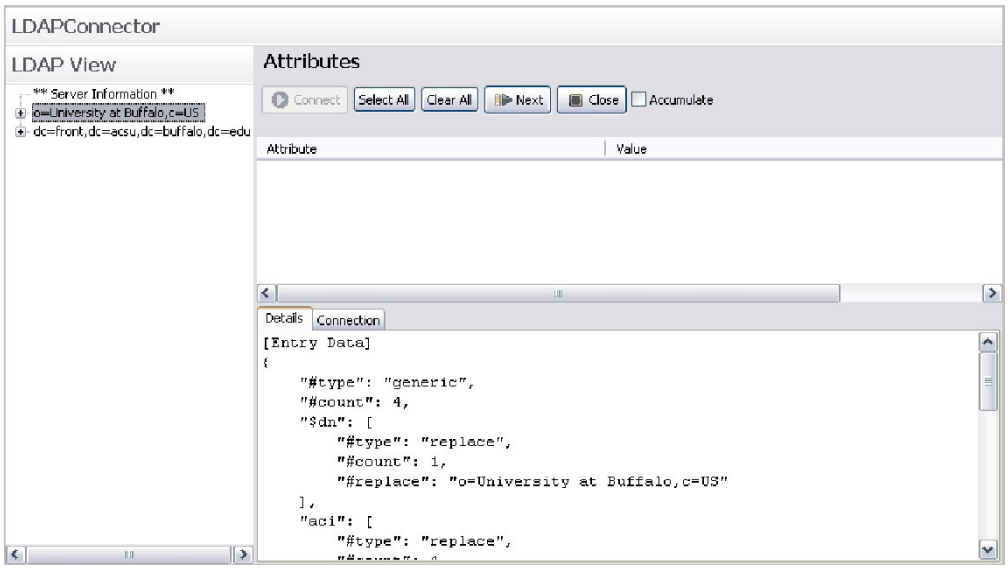


图 59. LDAP 数据浏览器条目

选择模式项时，您将在“详细信息”选项卡中看到详细信息，并使用模式项中的信息更新属性列表。如果您将要读取或编写特定模式，那么这非常有用:

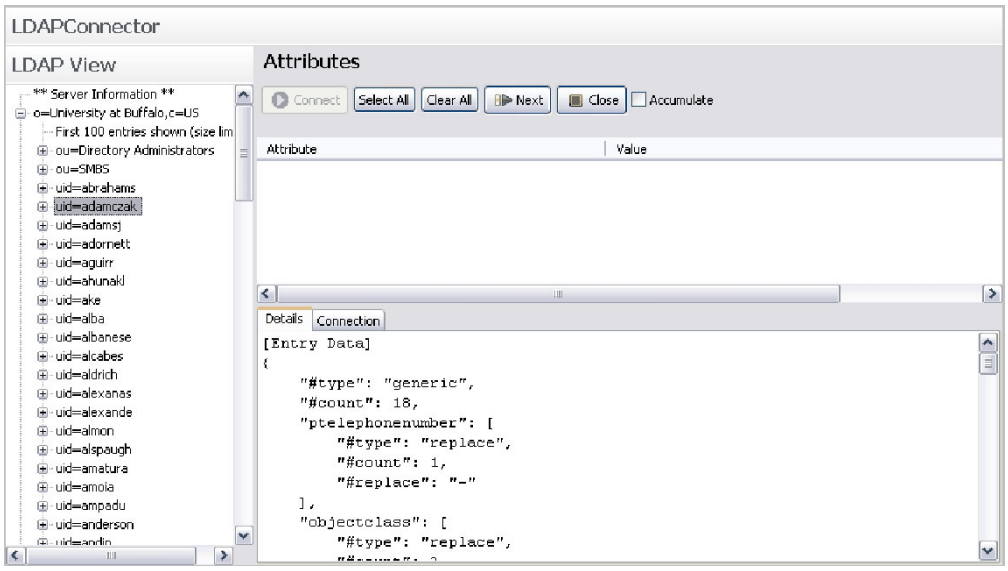


图 60. LDAP 数据浏览器模式项

在模式节点中选择对象类会让您快速为此类创建属性映射。现在，值列包含属性的有关信息。值“MAY”表示其可选，而“MUST”表示其为必需（在添加条目时）。括号中的值显示可定义属性的对象类；LDAP 对象类是分层的。

还可以通过从上下文菜单选择**用作搜索条件**来快速更新 LDAP 连接器的 **Search Base** 参数。

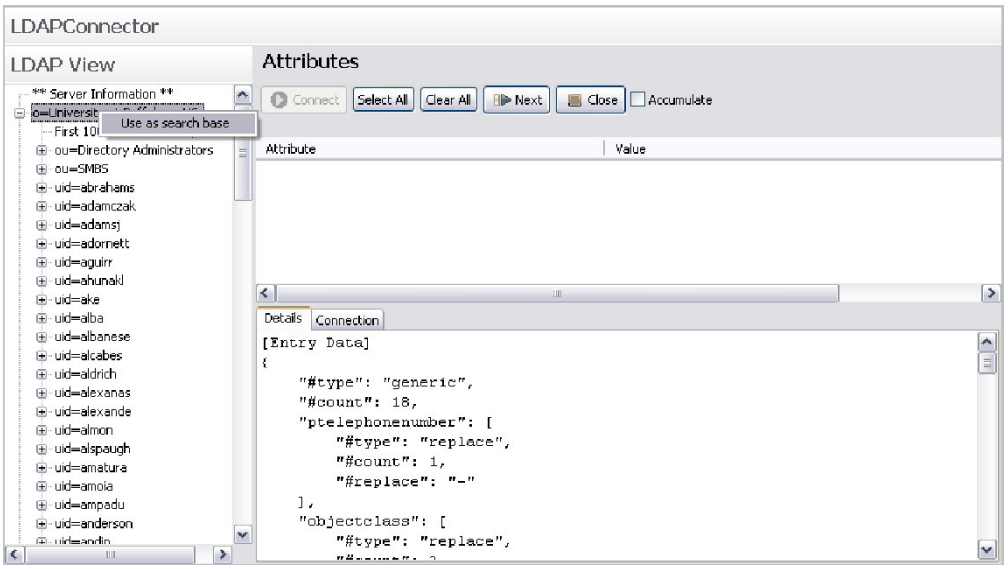


图 61. “用作搜索条件”上下文菜单选项

表单编辑器

表单编辑器用于定制组件的连接参数表单。这只能应用于 Resources 文件夹中的组件（除了属性文件）。

要定制表单，您必须用*表单编辑器*来打开组件。

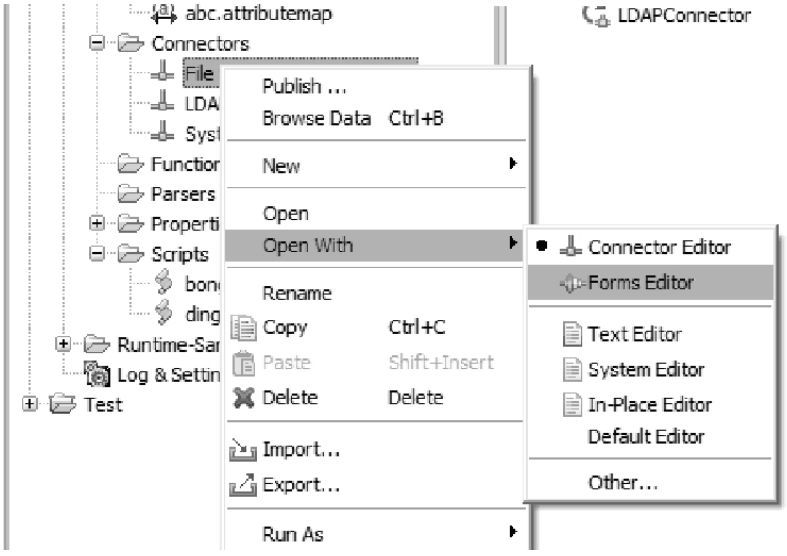
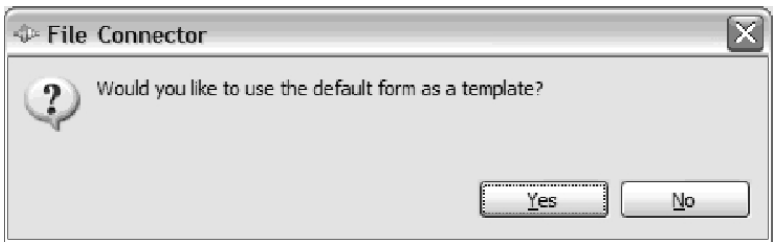


图 62. 上下文菜单 – 表单编辑器选项

请注意您为文件选择缺省值之外的其他编辑器时，CE 将记住您的选项，以便您下次双击此文件时，其将使用您上次用过的编辑器来打开此文件。要用缺省编辑器来打开组件，就选择此菜单中的适当编辑器（一般为最上方的编辑器）。

如果打开的组件没有定制表单，那么会提示您用缺省表单填充此表单：



选择**是**将根据组件的缺省表单创建初始表单定义。在此情况下，在此示例中使用 FileSystem 连接器，这样将生成以下屏幕:

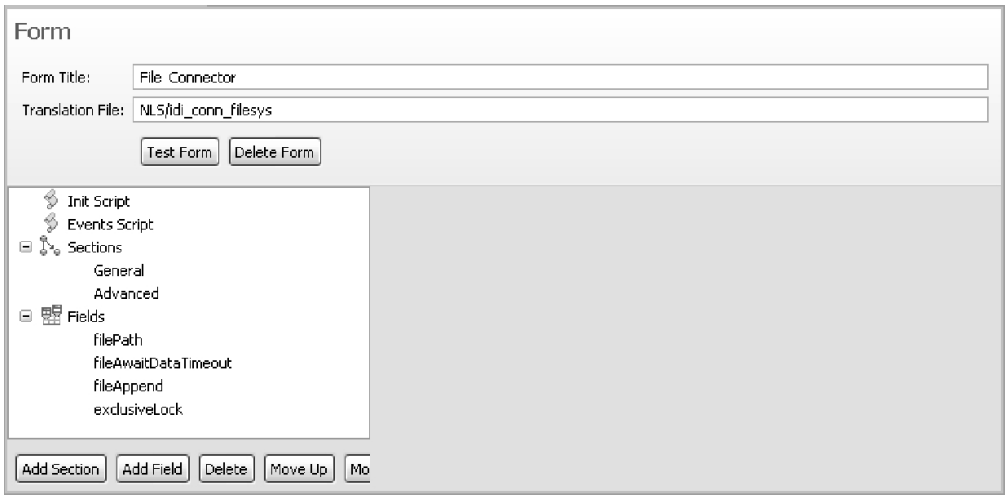


图 63. FileSystem 连接器的缺省表单编辑器屏幕

在此表单中看到的各种元素具有以下功能:

表单标题

这是组件表单的主标题（对于无标题的情况将留空）。这是用户在定制表单的顶部看到的内容。

翻译文件

这是用于翻译表单中标签和工具提示的文件。如果您定义了翻译文件，那么将尝试翻译所有标签和工具提示。如果您将带有“file_name”的标签创建为文本，那么翻译将尝试从使用“file_name”作为关键字的翻译文件检索字符串。翻译文件自身是在每行都带有“key=value”的简单属性文件。

要本地化表单，您必须用语言环境标识来创建文件。因此，对于法语翻译，您将创建称为 *base-name*_fr.properties 的文件。

测试表单

此按钮将显示带有当前表单定义的对话窗口。

删除表单

此按钮将从组件删除表单。您必须关闭并选择**保存**以使其永久生效。

初始脚本

装入表单时，将执行初始脚本。这是您放置代码的位置，以初始化表单的状态以及表单的任何全局脚本变量。

事件脚本

字段值更改时，表单将执行在此脚本中定义的事件处理程序。每个字段具有组件所用的内部名称。例如，FileConnector 使用“filePath”作为其 **File path** 参数的内部名称。您在“字段”部分选择用缺省表单填充表单时，可以看到所有组件参数名称。要对表单中的更改作出反应，您可使用以“_changed”为后缀的内部名称作为脚本函数名称来在事件脚本编辑器中编写事件处理程序。以下您会看到来自 LDAP 连接器的示例，其禁用了基于认证方法的两个字段:

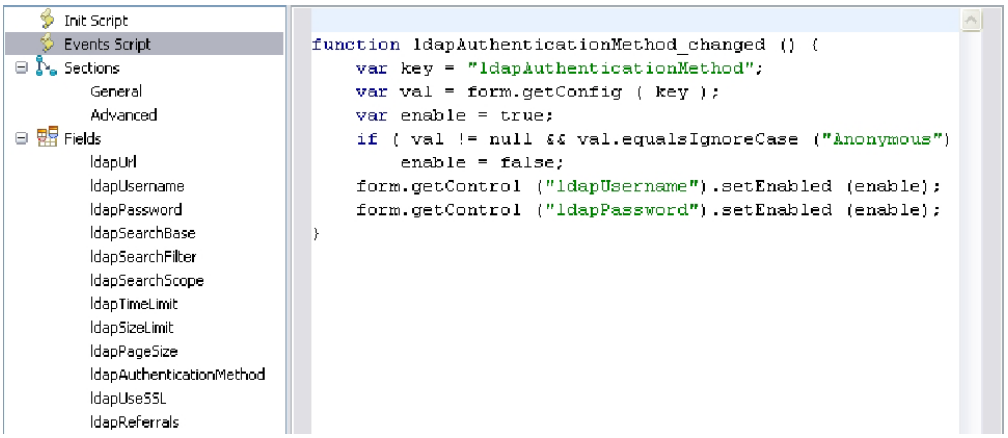


图 64. 表单编辑器，LDAP 连接器中的事件脚本

部分和字段

部分和字段组成了表单。您通过使用树下的工具栏来添加、除去部分和字段以及重新排列其顺序来管理部分和字段。

- **添加部分** – 向表单添加新的空部分
- **添加字段** – 向表单添加新字段。字段名称必须唯一。
- **删除** – 从表单除去某个部分或字段。
- **上移/下移** – 重新排列各个部分和表单的顺序。定义了各个部分时，由此部分（而不是由字段的列表）定义字段的顺序。未定义任何部分时，此树中的顺序决定表单中的字段顺序。

部分 此部分可选。如果您没有指定部分，那么表单将在其表单中一次性显示所有字段。

各部分用于排列表单中的字段。各部分如同文件夹，用户可以将其展开和折叠以显示/隐藏其内容。定义某部分时，您指定初始是否展开此部分以及在其中要显示哪些字段。在 FileSystem 连接器示例中，有两个部分。

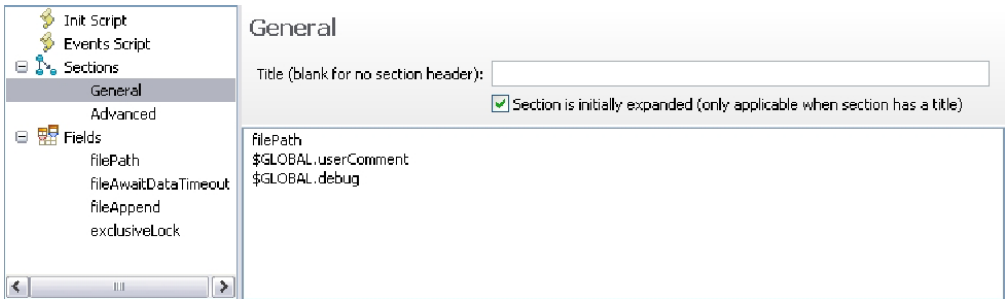
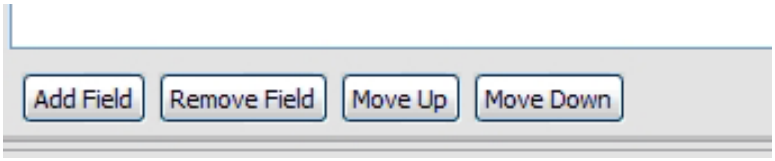


图 65. 表单编辑器 – 常规部分

第一个部分是*常规*部分。此部分没有标题，这意味着没有用户可以单击以展开或折叠内容的部分标题。由于无法折叠或展开此部分，所以这使该部分成为静态部分。您可以使用面板底部的工具栏来在字段列表中添加、除去字段并重新排列其顺序:



第二部分是*高级*部分:

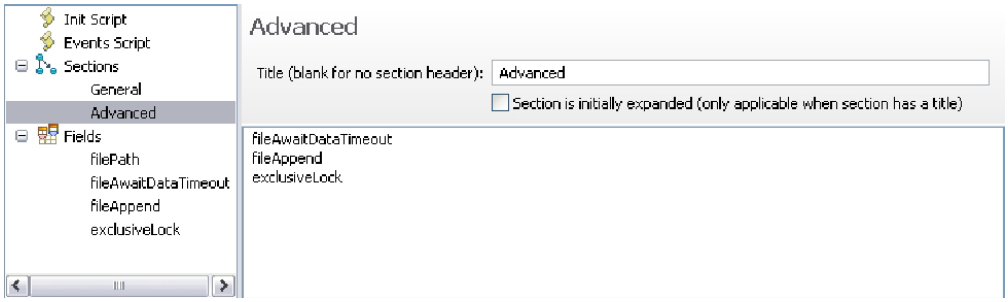


图 66. 表单编辑器 – 高级部分

此部分有标题，但初始时其并未展开。这将使表单显示折叠了此部分标题的表单，且隐藏了其中的字段，直到用户展开此部分为止。

字段 字段是组件的参数。如果您复用诸如 FileSystem 连接器之类的组件，那么您应该在表单中提供必需参数或在“初始脚本”部分中设置参数，以便组件可正常发挥功能。每个字段具有您将其选中时所示的定义。

图 67. 表单编辑器 – 字段定义

在此面板中，您可看到连接器参数“filePath”的定义。在显示顺序中:

表 9. 表单编辑器 – 参数定义

字段	描述
标签	表单将显示的标签
工具提示	用户鼠标置于输入字段上时显示的工具提示
字段类型	输入字段的类型: • 字符串 用于单行文本输入 • 下拉（可编辑） 用于可编辑的下拉输入字段 • 下拉（不可编辑） 用于具有固定选项集的下拉项 • 布尔 用于复选框 • 文本区域 用于多行文本输入字段 • 静态文本 用于简单文本显示（即无输入） • 密码 用于单行密码保护输入字段 • 脚本编辑器 用于编辑脚本 • 定制组件 用于用户定义的 SWT/JFace 控制
方式选择	此可选字段可指定在其中排除或包含此字段的组件方式。以负号来指定要排除的方式（以逗号分隔）。 “Iterator” – 仅以 Iterator 方式显示 “-Iterator” – 不以 Iterator 方式显示 “Iterator,Lookup” – 仅以 Iterator 和 Lookup 方式显示

底部的三个选项卡可让您指定下拉列表的按钮和下拉值，以及定制组件的 Java 类名称。

向导

IBM Security Directory Integrator 配置编辑器中存在若干向导（图形化辅助分步过程）。它们是:

1. 第 125 页的『“导入配置”向导』
2. 第 127 页的『“新建组件”向导』

3. 第 133 页的『连接器配置表单特性』

“导入配置”向导

您可以使用“导入配置”向导来导入先前版本的配置文件。

在此向导中，选择目标项目和要导入的组件。

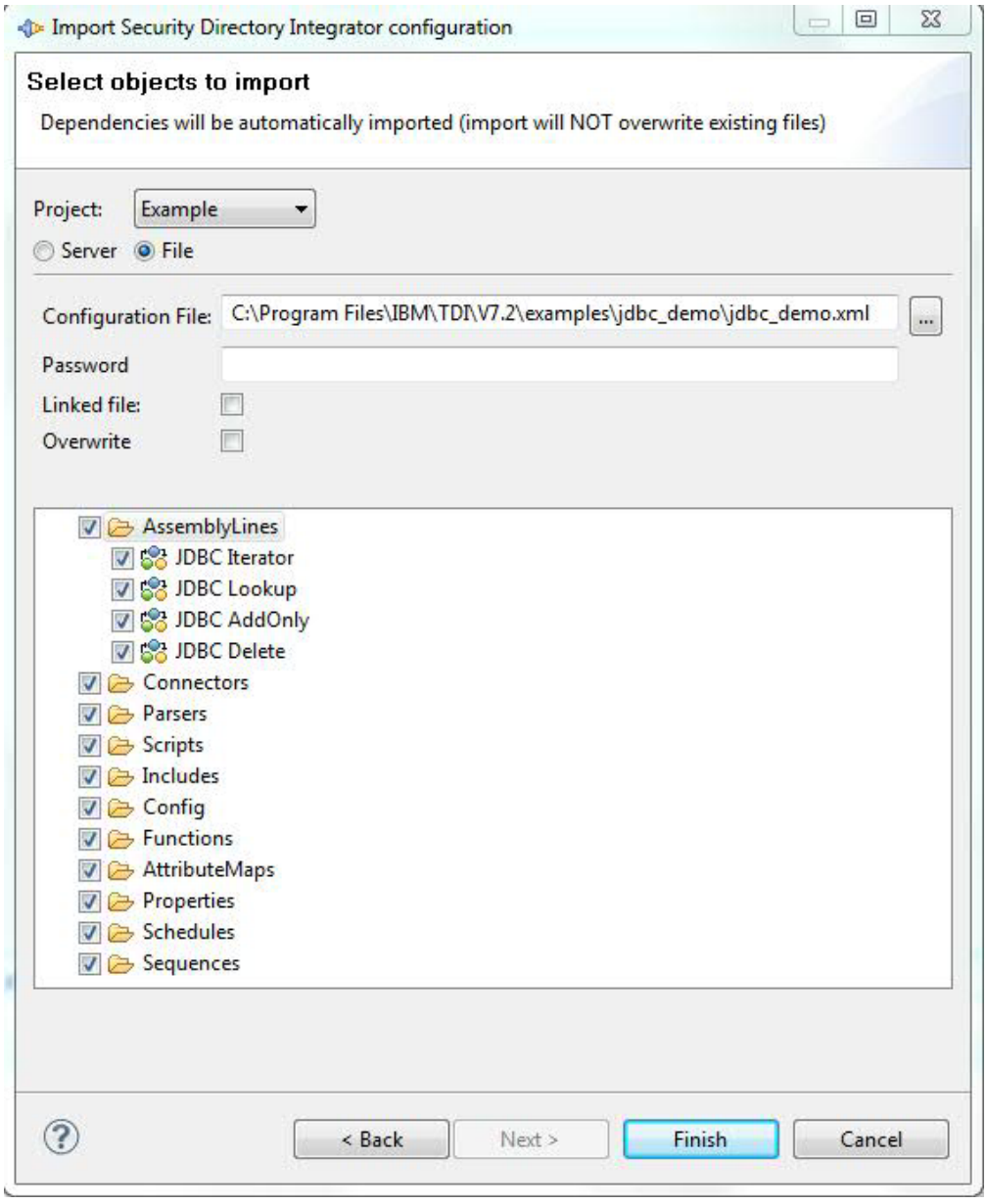


图 68. “导入配置”向导

缺省情况下，将选择所有组件进行导入。但是，您可以只选中您感兴趣的那些组件。如果您选中了一个 AssemblyLine，且该 AssemblyLine 使用配置文件中的连接器，那么也将自动导入那些连接器。

项目输入字段是要导入配置的目标项目。选择空白选项可创建新项目。

配置文件输入字段是您即将导入的配置文件。如果配置是受密码保护的，那么在**密码**输入字段中输入密码。

如果选中了**链接文件**字段，那么对已导入项目进行的任何更改都会回写到导入项目所通过的文件中。通过清除或更改**链接文件**字段的文件名，可以在项目属性中更改此设置，如下所示:

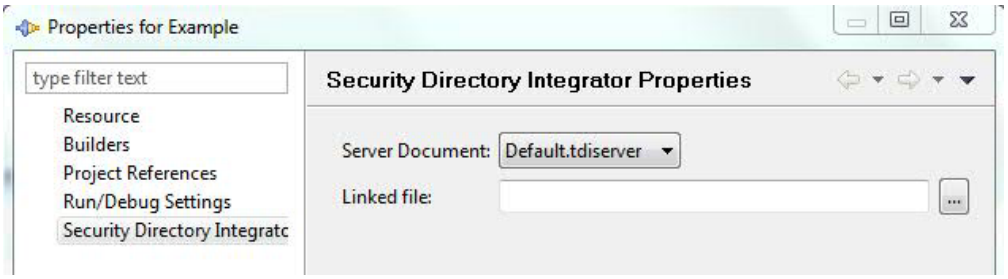


图 69. “链接文件”字段

您还可以从服务器导入配置。通过选中**服务器**单选按钮，可以切换到服务器视图。

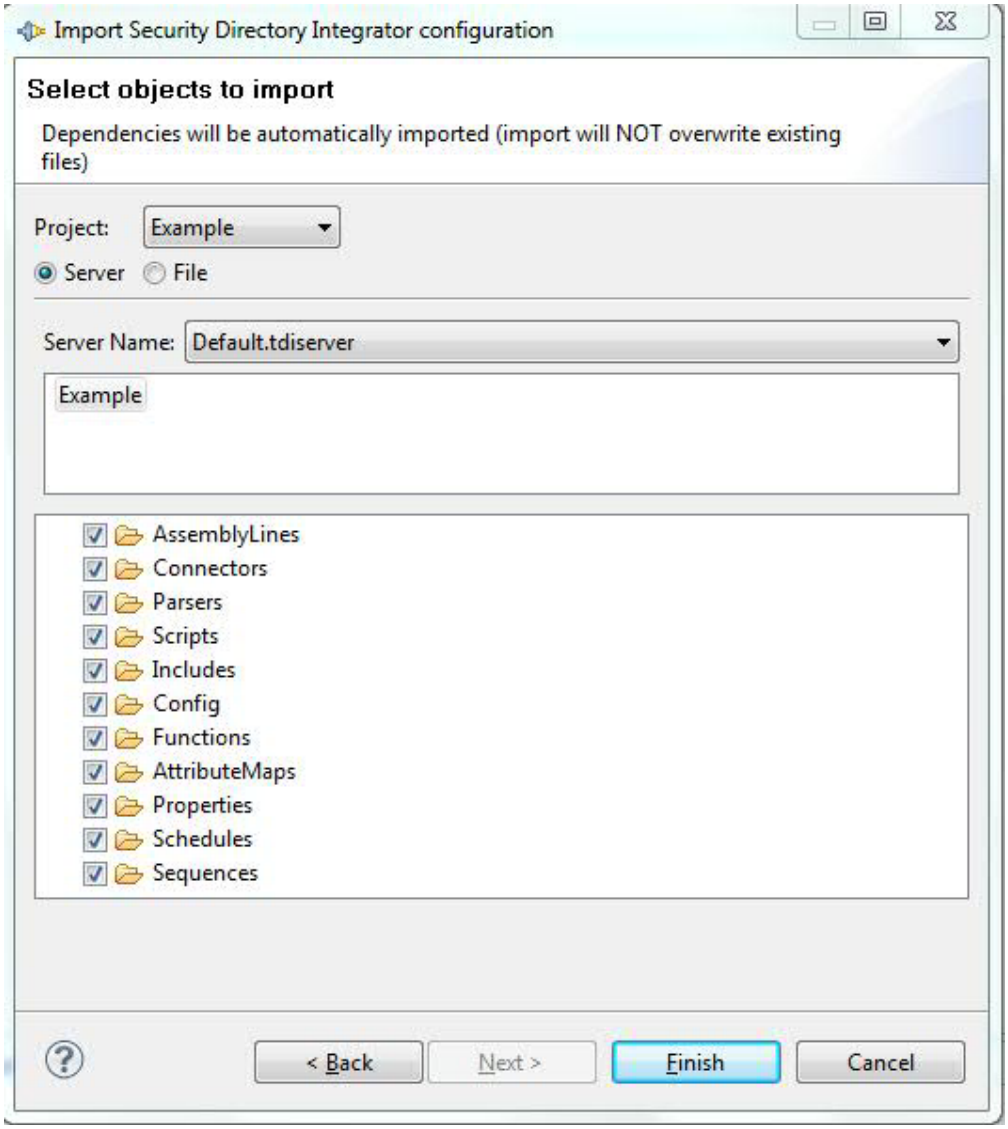


图 70. “从服务器导入”向导

通过从**服务器名称**字段中的服务器列表选择服务器，可以启动此类型的导入。一旦选择了服务器，便会在服务器名称下方的列表中显示该服务器的配置列表。由于这是网络操作，因此在刷新列表时，部分控件可能会显示为已禁用。从配置列表中，可以选择要导入的配置。选择某个配置将会从服务器下载该配置，并且使用其内容填充下方的树，从而可以选择要在导入中包含的组件。

“新建组件”向导

当您在 AssemblyLine 中使用**添加组件**或当您使用主菜单栏的**文件 > 新建...**时会调用此向导。

当您在 Resources 文件夹中创建新组件时，向导的布局将稍微有所不同。例如，对于新的连接器，向导看起来类似于:

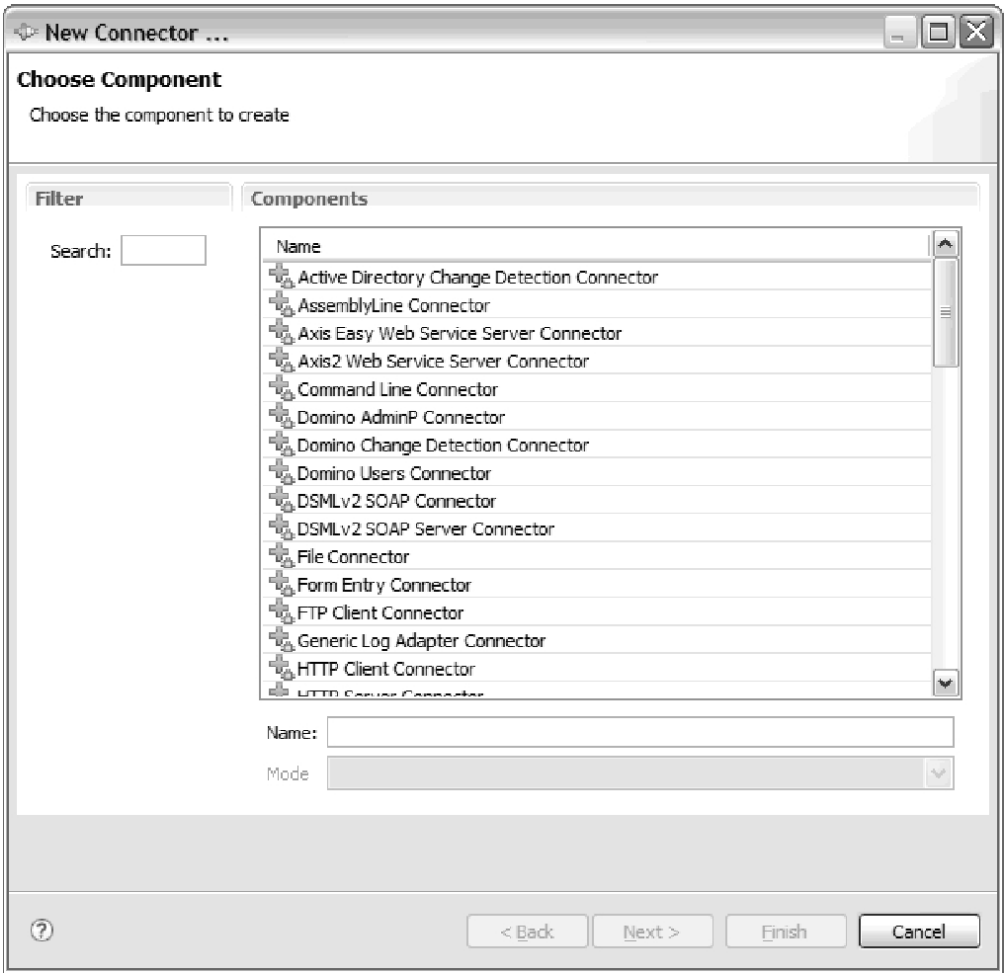


图 71. Resources 文件夹向导中的新连接器

组件名称将用作文件夹中建议的文件名；将该名称更改为有意义的名称是明智的。

当您在 AssemblyLine 中创建新的组件时，您将在此向导中具有更多选项。

向导的第一个页面是组件类型选择页面。在该页面中，可以选择您希望添加或创建的组件类型。左边包含将根据列表中的标签选择相关组件的过滤器列表。

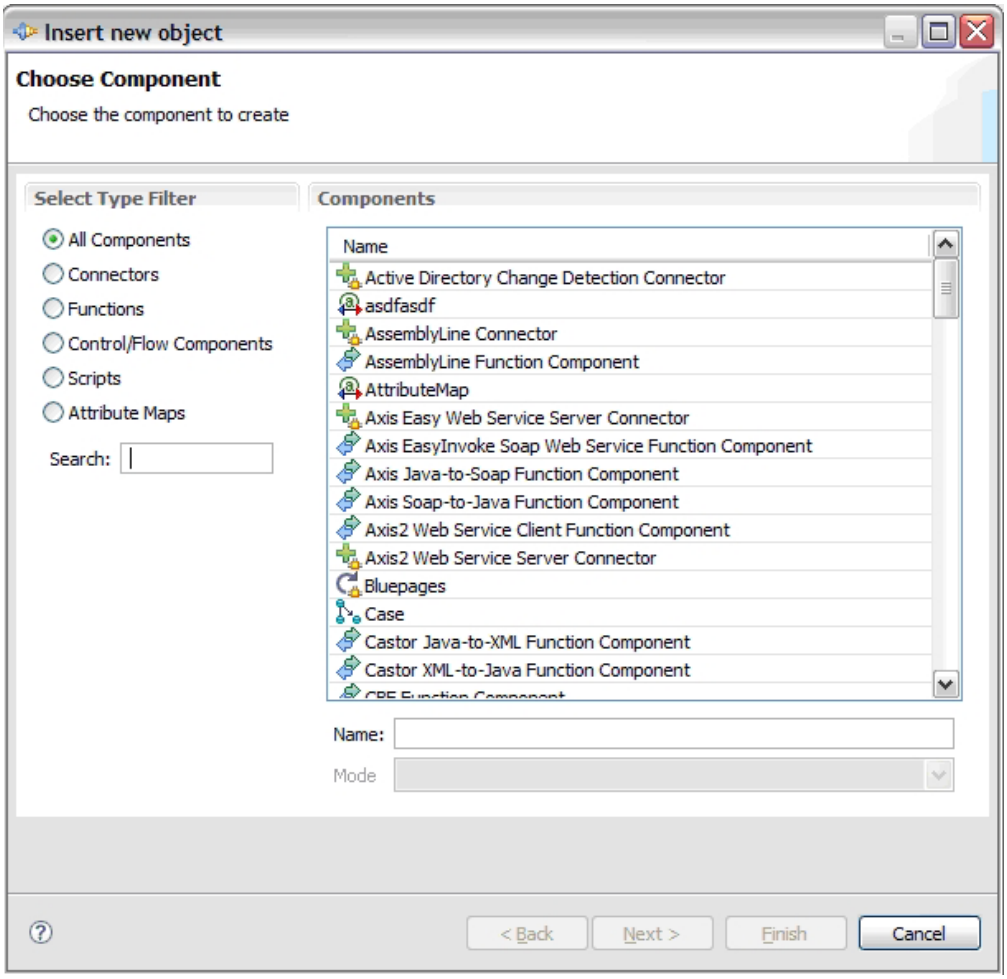


图 72. “新建组件”向导

您可以通过在“搜索”字段中输入来过滤内容。当您输入时，将针对您在搜索字段中输入的内容对列表进行匹配（匹配不区分大小写）。

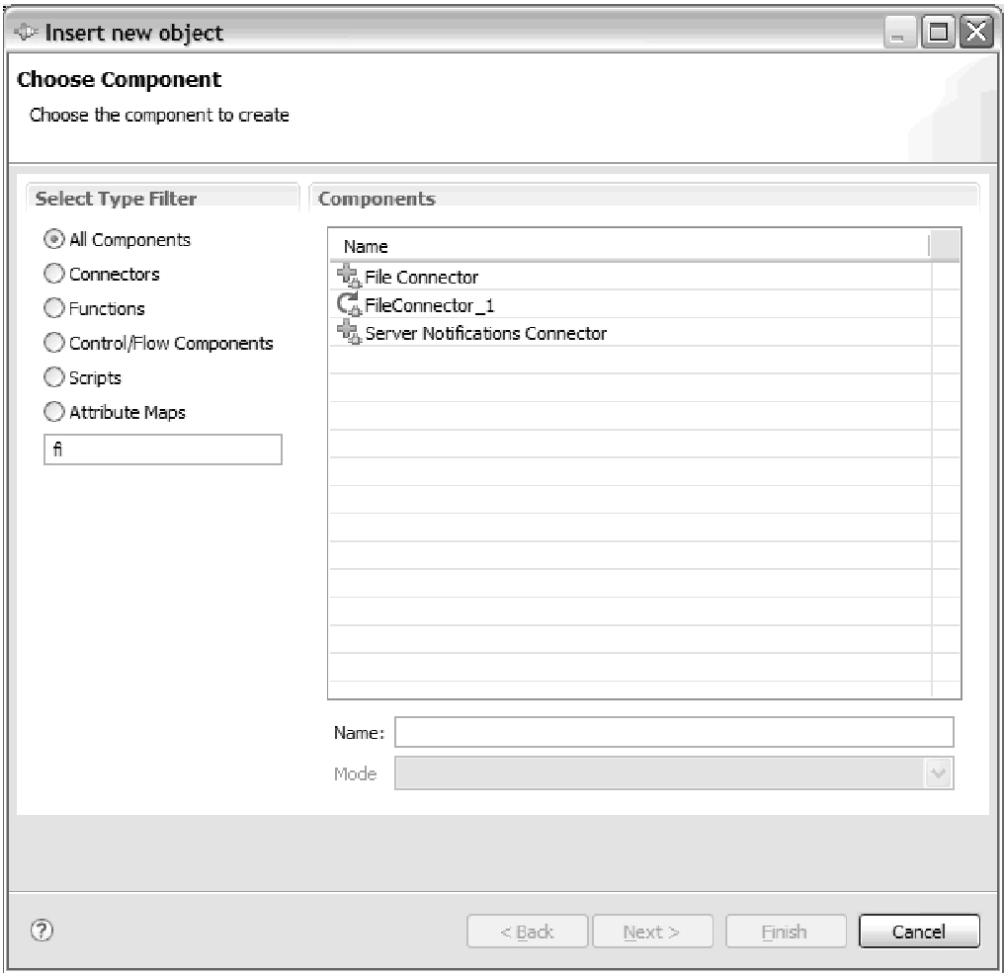


图 73. "新建组件"向导，使用过滤

在这里，您可以选择完成向导，组件将插入到您的 AssemblyLine 中。

选择连接器时，您将看到用于正确定义连接器的一系列表单。对于所有其他类型，您只能在 AssemblyLine 中完成向导并配置组件。

选择类型后，您将看到"连接器配置"面板。

图 74. “连接器配置”面板

另请参阅第 133 页的『连接器配置表单特性』了解完成此类型表单的详细说明。

如果连接器可以使用解析器，下一步将显示“解析器”配置。

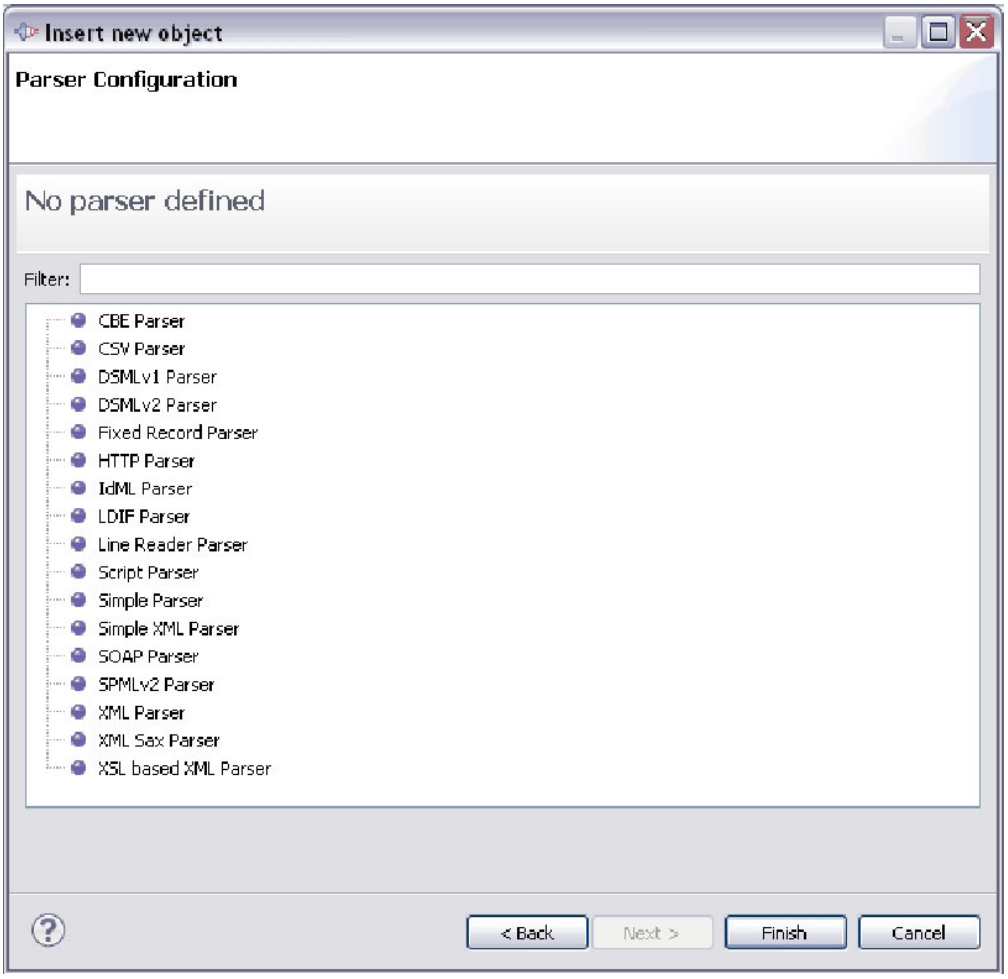


图 75. “解析器配置”面板

使用**选择解析器**按钮可为组件选择解析器:

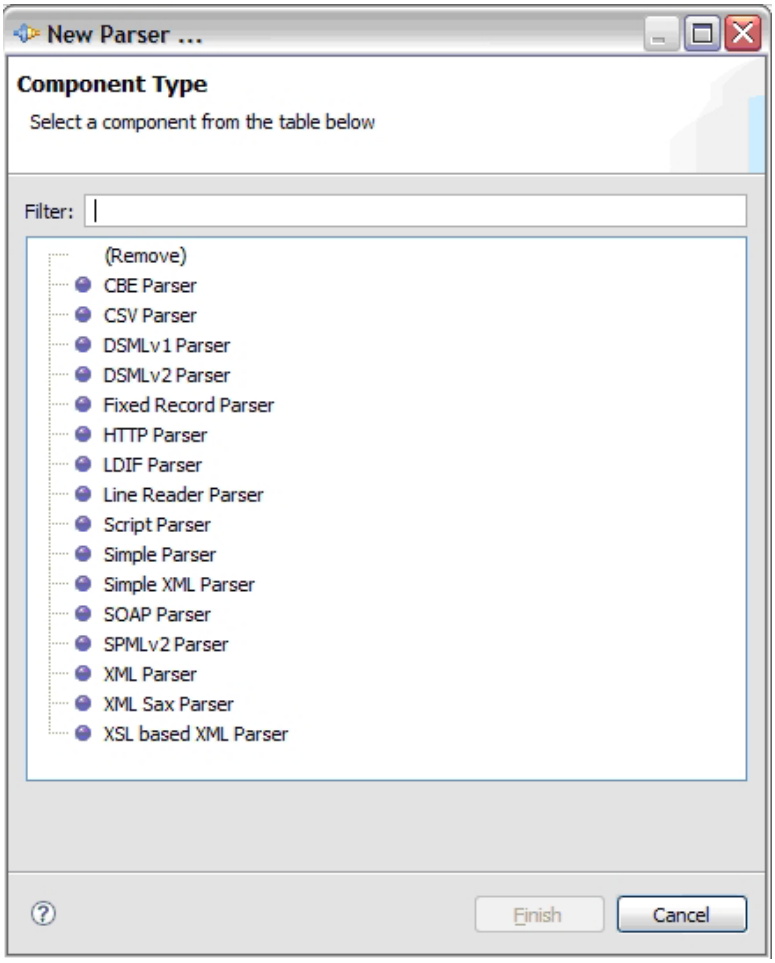


图 76. 解析器选择对话框

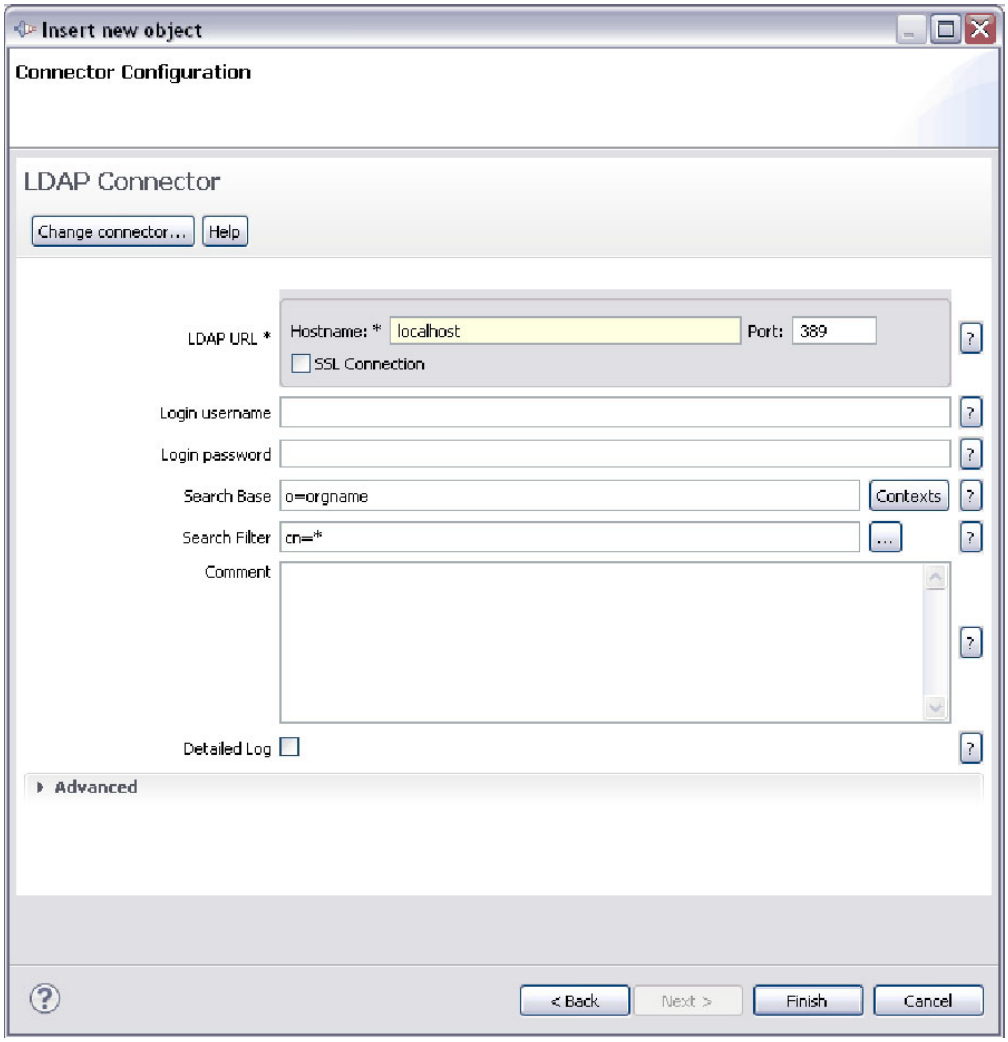
您可以通过选择“（除去）”选项从组件中除去当前解析器。

连接器配置表单特性

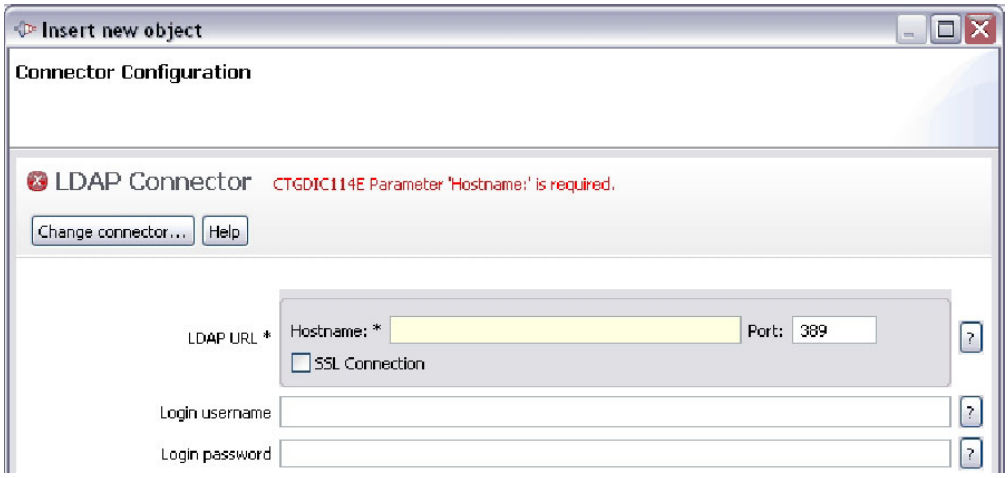
表单中的字段可以选择性地分组至有标题的部分，且这些字段具有指定字段是否必需的属性。

在“连接器配置”窗口中，必填字段由字段名称后的 * 来表示。

被继承的值具有蓝色标签。



在以上样本中，LDAP 连接器配置显示具有必填字段的两个部分。必填字段无值时，表单将在标题中标记该字段。



将鼠标移至红色图标或文本上时，将看到相关的错误消息，以及这些错误对应的字段。当表单中有多个错误时，这种方法很有用。

运行和调试 AssemblyLine

配置编辑器中提供了多个机制，这些机制可帮助您开发 AssemblyLine（包括帮助测试和调试 AssemblyLine 的逻辑）。

AssemblyLine 报告

可以从导航器中的上下文菜单中运行 AssemblyLine 报告。

右键单击组装流水线并选择**创建组装流水线报告**子菜单。此菜单包含位于 *TDI_Install_dir*/XSLT/ConfigReport 目录中的所有报告模板。

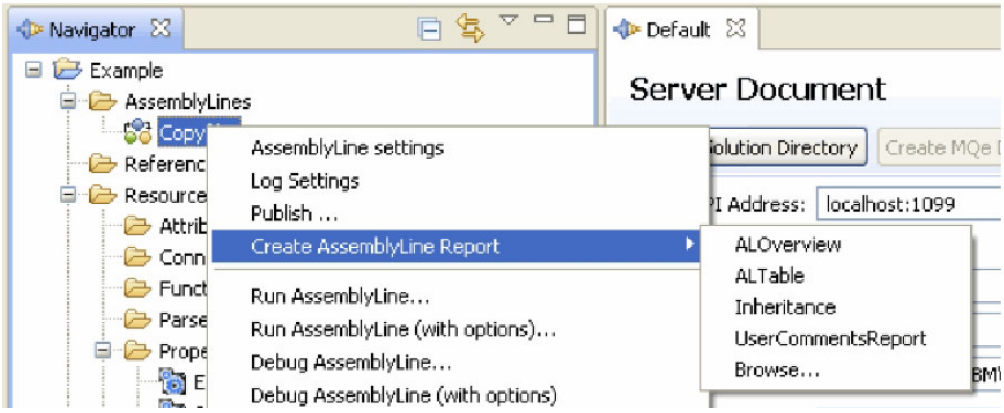


图 77. “创建组装流水线报告”命令

选择**浏览...**选项以浏览报告模板的本地文件系统。

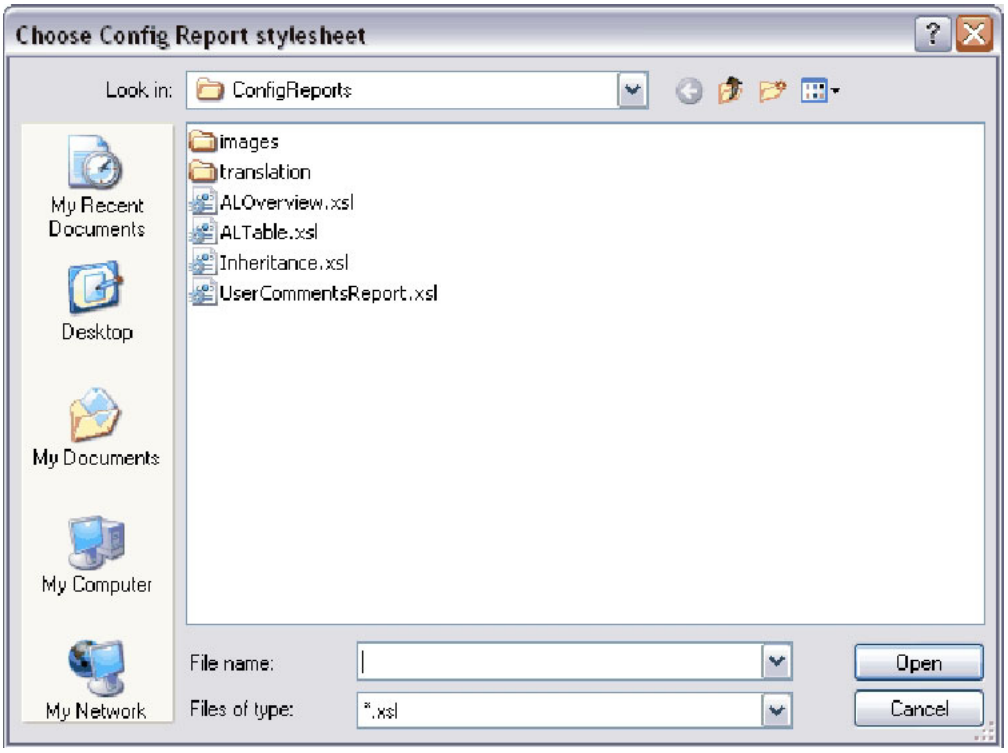


图 78. “选择配置报告样式表”对话框

当选择一个文件并单击**打开**时，将生成报告，且该报告将置于如下图所见的项目的 Reports 目录中。将打开与 *.html* 文件扩展名相关联的编辑器以查看报告，通常是缺省的因特网浏览器。

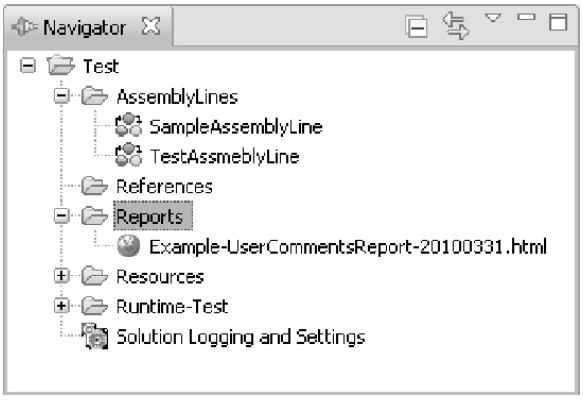
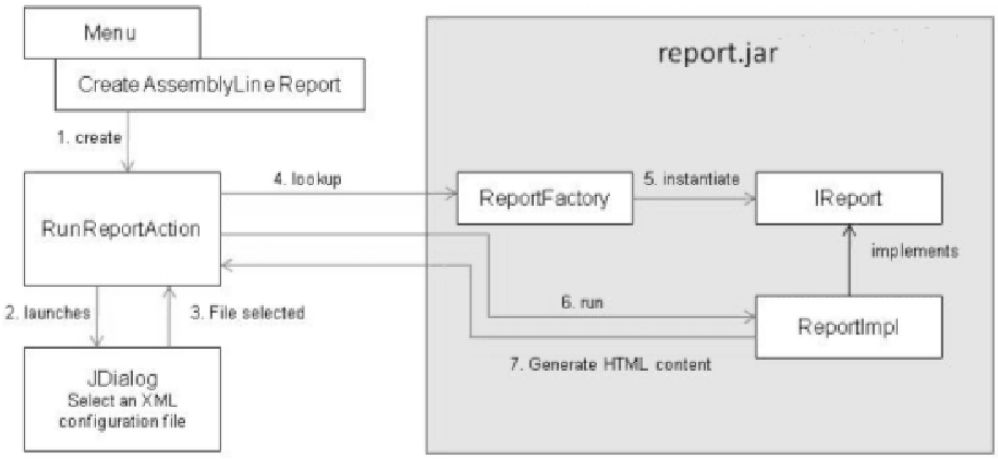


图 79. 项目层次结构中的报告文件夹

组装流水线报告根据组装流水线生成报告文件名，并插入当前日期。

基于 XML 的组装流水线报告概述

可以基于报告样式表为所选配置元素生成报告，也可以为指定的报告配置 XML 文件生成报告。下图说明了基于 XML 的 AssemblyLine 报告的体系结构。



报告配置 XML 文件格式

组装流水线报告配置 XML 文件的格式如下:

```
<tdiReport>
<reportClass)com.ibm.di.report.aloverview.AssemblyLineOverview</reportClass>
    <reportConfig>
        <report specific configuration>
    </reportConfig>
</ tdiReport>
```

配置 XML 文件有下列元素:

- **reportClass** - 指定报告的 Java 类名称。
- **ReportFactory** - 启动报告的实例。
- **reportConfig** - 包含报告特定参数。
-

运行 AssemblyLine

开发 AssemblyLine 时，可以通过运行至完成或逐步逐个运行组件来测试该 AssemblyLine。

有两个运行 AssemblyLine 的按钮。第一个按钮（播放图标， ）用于启动 AssemblyLine，并在控制台视图中显示输出。第二个按钮（**调试器**）用于通过调试器运行 AssemblyLine。

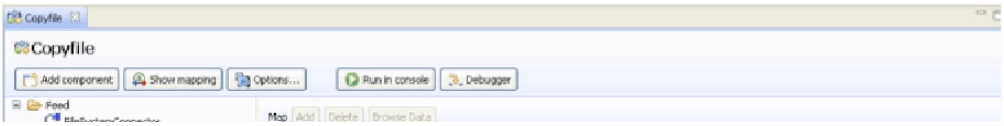
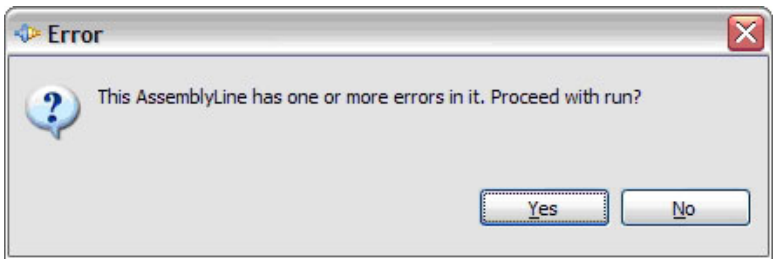


图 80. 用于启动组装流水线的三个选项

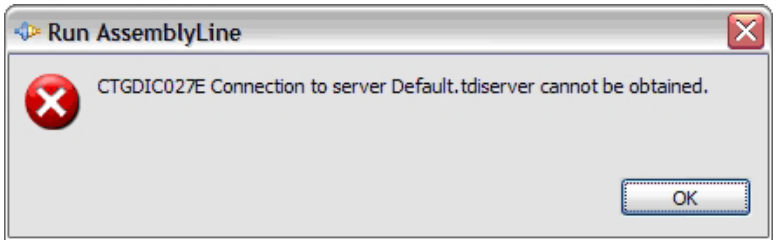
启动 AssemblyLine 的进程执行这三个步骤。

如果 AssemblyLine 包含错误（例如缺少输出映射等），那么系统将提示您确认运行 AssemblyLine:

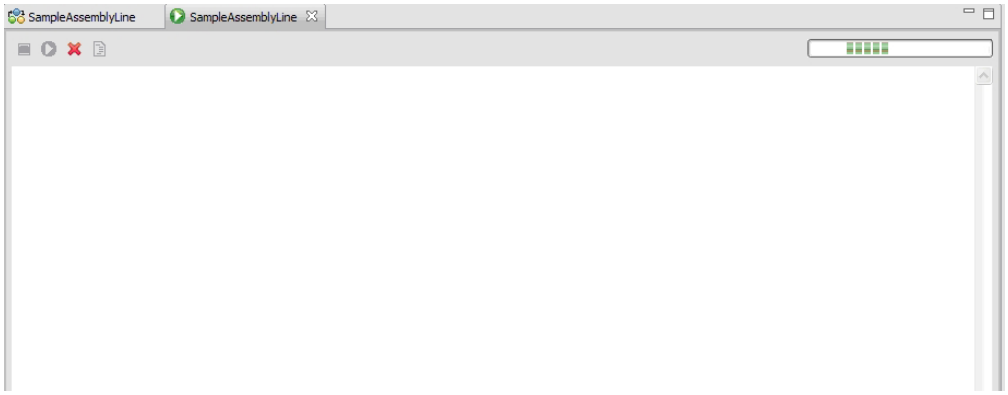


如果看到该对话框，应该检查“问题”视图以查看哪些错误可能将要中断 AssemblyLine。通常情况下，在开发期间尽管您意识到这些问题却仍然想要运行 AssemblyLine，如果那样，请按 Enter 或单击**是**按钮以运行 AssemblyLine。

接下来检查 IBM Security Directory Integrator 服务器是否可用。如果服务器不可访问，您将看到该消息:

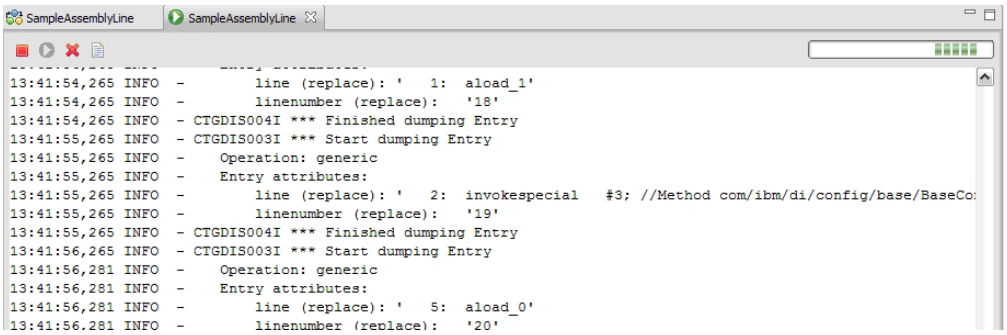


当从 CE 中运行 AssemblyLine 时，第一步是 CE 将运行时配置传输至服务器并等待 AssemblyLine 启动。在此步骤中，将在窗口右上角部分中看到进度条。停止 AssemblyLine 的工具栏按钮也变灰，因为其尚未启动。



第二步在运行 AssemblyLine 时执行。进度条将滚动并且应该开始在日志窗口中查看消息。此时通过单击工具栏中的停止按钮（最左侧）可以停止 AssemblyLine。

注: 仅当服务器获得了对线程执行的控制权时该停止按钮才有效。如果该线程正在执行 IBM Security Directory Integrator 之外的某些操作，那么单击停止按钮可能不起作用。



如果具有多个打开的 AssemblyLine 窗口，那么可以了解哪些对应的 AssemblyLine 正在运行，因为其名称加有“*”（星号）前缀。

如果 AssemblyLine 已停止（要么正常停止，要么单击停止按钮），那么进度条将消失，且将启用工具栏项以重新运行 AssemblyLine。此时停止按钮已禁用，因为 AssemblyLine 不再运行。

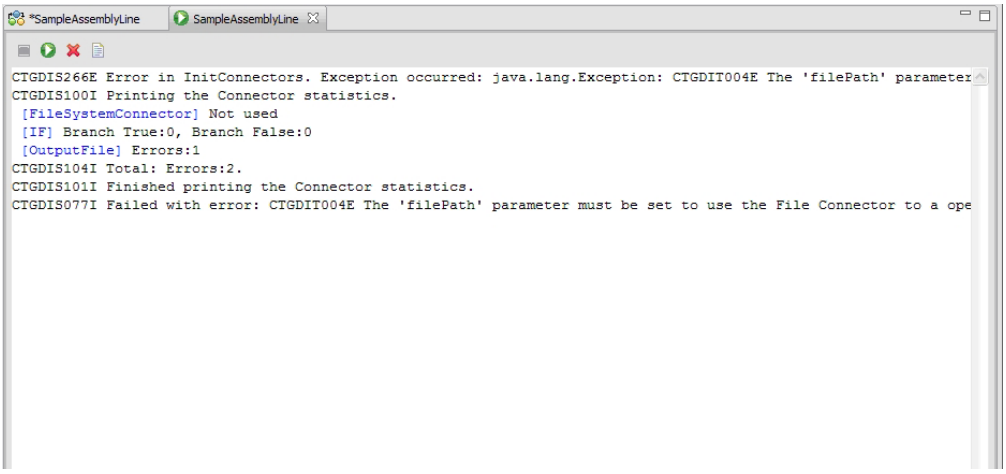


图 81. 控制台日志窗口

可以随时清空日志窗口。日志窗口仅显示 AssemblyLine 日志的最后几百行，但是每条日志消息都将写入临时日志文件，以便可以使用工具栏（最右侧）中的“查看”日志按钮在单独的编辑器窗口中打开日志文件。

注：您可以将日志窗口底层的日志缓冲区大小从缺省的 300 行更改为其他值，方法是转至**窗口 > 首选项 > Security Directory Integrator 首选项 > “运行 AssemblyLine”窗口的的最大行数**。

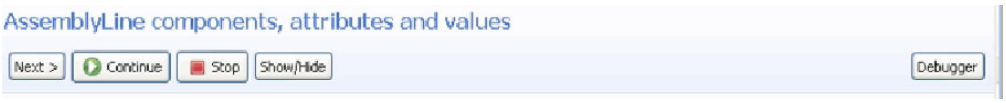
一旦 AssemblyLine 终止，可以使用重新运行按钮来重新运行 AssemblyLine。

请注意日志窗口中的蓝色文本。当您看到该文本时，单击该文字时按住 CTRL 键可引导您浏览 AssemblyLine 的该部分。

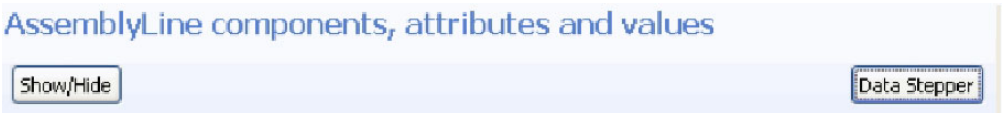
步进器和调试器

步进器和调试器是配置编辑器内置的工具，可用于帮助您以交互方式开发 AcessemblyLine。

可以在两种方式下运行步进器。一种是一般高级调试器（通过**调试器**按钮激活，请参阅第 141 页的『调试器』），您可以在其中访问 AssemblyLine 的所有部分；另一种是步进器（通过**数据步进器**按钮激活，请参阅『数据步进器』），它提供更简单的组件和流程视图。您可以通过单击列视图中的按钮在两种方式之间切换:



在步进器视图中，可以通过单击**调试器**按钮切换到调试器视图。反之，单击**数据步进器**按钮可以从调试器切换到步进器。



数据步进器

数据步进器提供 AssemblyLine 中所有连接器的列视图。逐步查看 AssemblyLine 将看到针对各组件读/写的数据。这些表中的所有数据总是在连接器完成其操作后显示。

图 82. 数据步进器主窗口

在右边部分中，数据步进器垂直显示带有属性映射的组件。通过单击关闭按钮或在**显示/隐藏**对话框中取消选择各组件，可以将其从视图中除去。各组件中关闭按钮左侧的按钮（  ）是“运行至此处”的快捷键。

下一个和**运行**按钮用于每次步进一个组件或运行 AssemblyLine 直至完成。**停止**按钮用于暂停运行中的 AssemblyLine 或终止已暂停的 AssemblyLine。数据步进器的左边部分显示 AssemblyLine 和下方工作条目的大纲。在大纲视图中，可以选择从上下文菜单重新启动 AssemblyLine。这将在新会话中启动调试器并运行至选定的组件。这是从数据步进器进入调试器的一种快速方法。

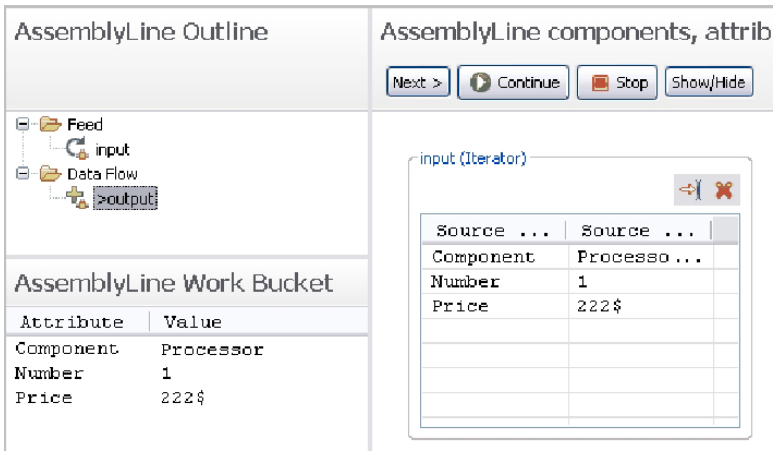


*图 83. 数据步进器中的**显示/隐藏**按钮*

通过**显示/隐藏**组件对话框，可以选择要在视图中显示的组件。

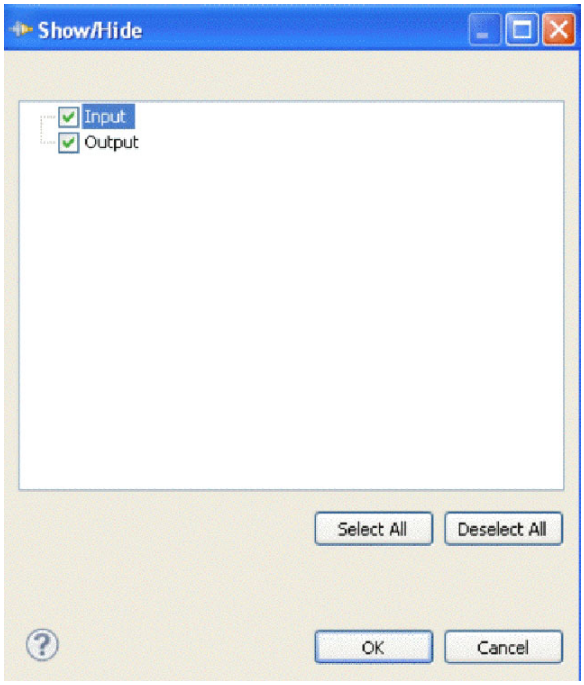


图 84. “显示/隐藏”组件对话框

调试器

如果选择**调试器**视图，您将会看到与步进器视图非常类似的布局，但是在调试器视图中，你看到的更多是 AssemblyLine 组件，此外还会看到监视窗口，其中可包含定制表达式。组装流水线组件树含有与各项对应的复选框，您可以将其选中或取消选中来设置或除去断点。此外，还有更多命令按钮，用于支持步入组件和挂钩。

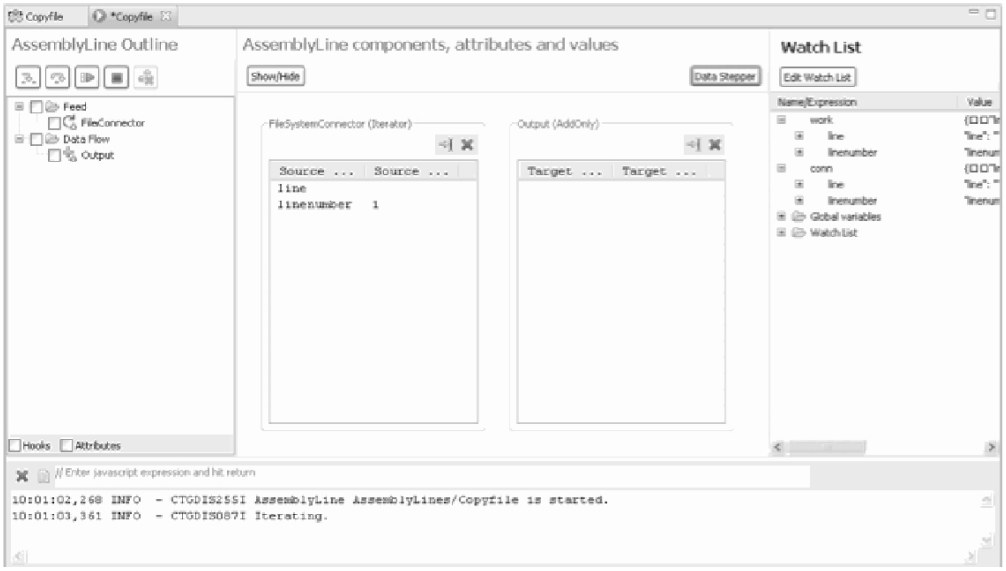


图 85. 调试器窗口

（在调试器窗口中，可用于使用 Ctrl-M 来最大化编辑器以获得更好的概述。）

左侧的树显示 AssemblyLine 组件和挂钩。可以切换每一项的复选框以在该处设置断点。双击该项以查看脚本，或输入条件中断的脚本。

下图显示了双击 **Before GetNext** 挂钩之后的条件中断选项卡。并且请注意，您可以隐藏不活动的所有挂钩。显示所有挂钩会让您不考虑挂钩是否活动而设置断点。**属性**复选框使您隐藏树形视图中的属性映射。

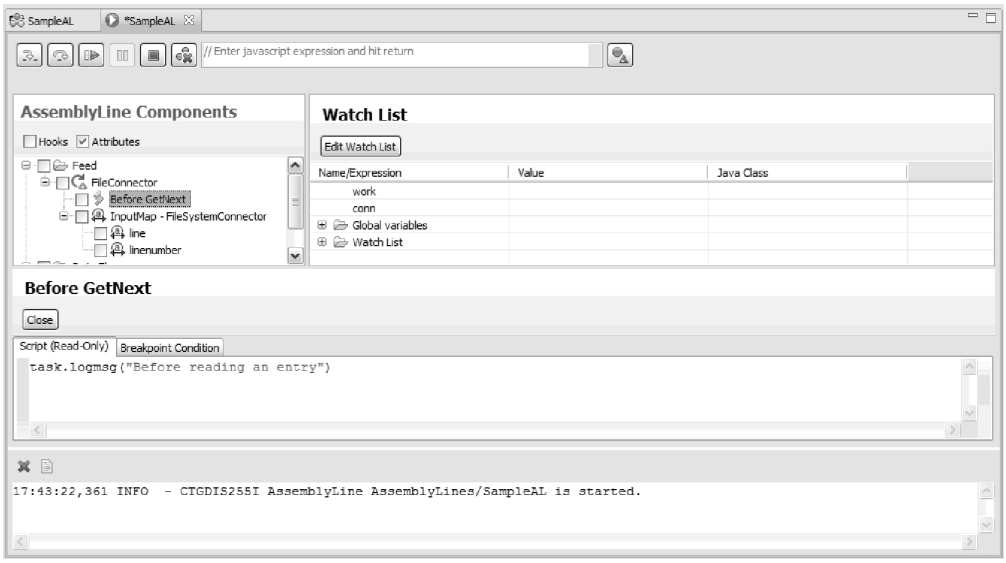


图 86. Before GetNext 位置上的调试器

属性映射面板显示组件及其指定，还显示指定给工作属性的值。该值是 AssemblyLine 中上次中断的快照，**先前值**是该快照之前的快照值。逐步执行 AssemblyLine 时，该值将反映影响工作条目的映射和脚本。

发生错误时，步进器将在单独的对话框中显示异常消息和堆栈跟踪。日志文件（屏幕上）不包含该堆栈跟踪。可以通过选择该复选框来选择不显示该对话框，或将该对话框显示在配置编辑器的首选项页面中。

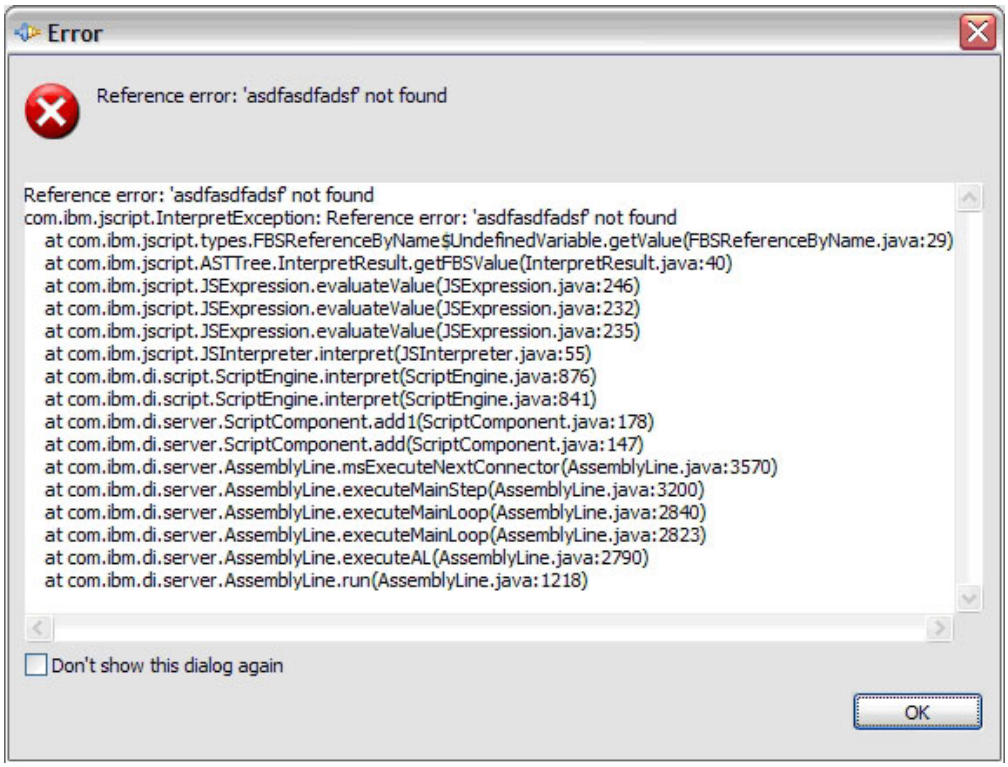


图 87. “错误”对话框：堆栈跟踪

逐步执行脚本

到达包含 JavaScript 的断点时，可以步入脚本中并逐行执行该脚本。该脚本选项卡将显示即将执行的脚本；可以使用 **Step Into** 命令（两个步进器按钮中最左边的一个）来执行该流程。

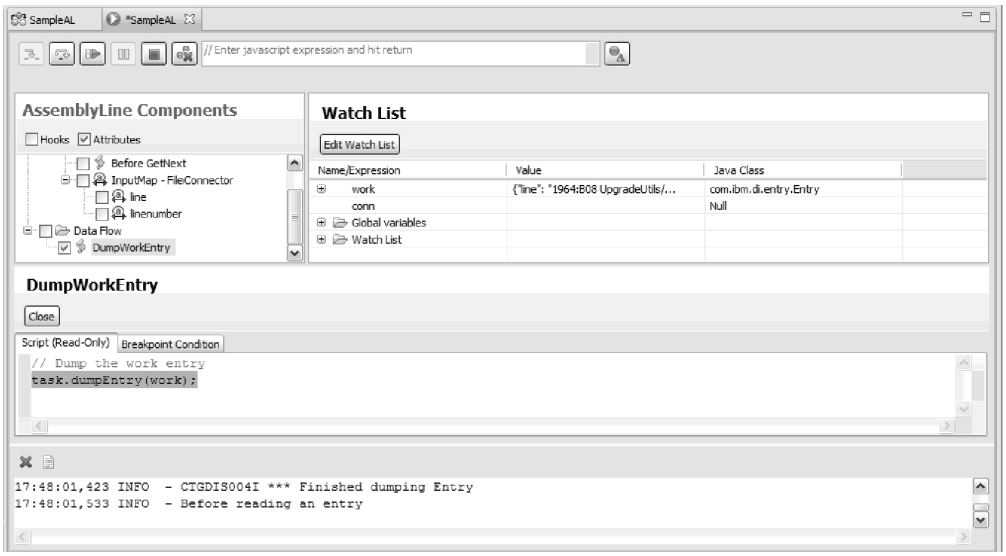


图 88. 调试器窗口：逐行对脚本进行单步调试。

逐步执行脚本时，每一行在执行之前都将突出显示。逐步执行脚本时，还可以使用**评估**功能以显示脚本引擎变量。

即将执行脚本函数时，可以使用 **StepInto** 按钮来步入 JavaScript 函数。如果函数在当前上下文（例如挂钩和属性映射脚本）外部进行定义，那么会替换该窗口。

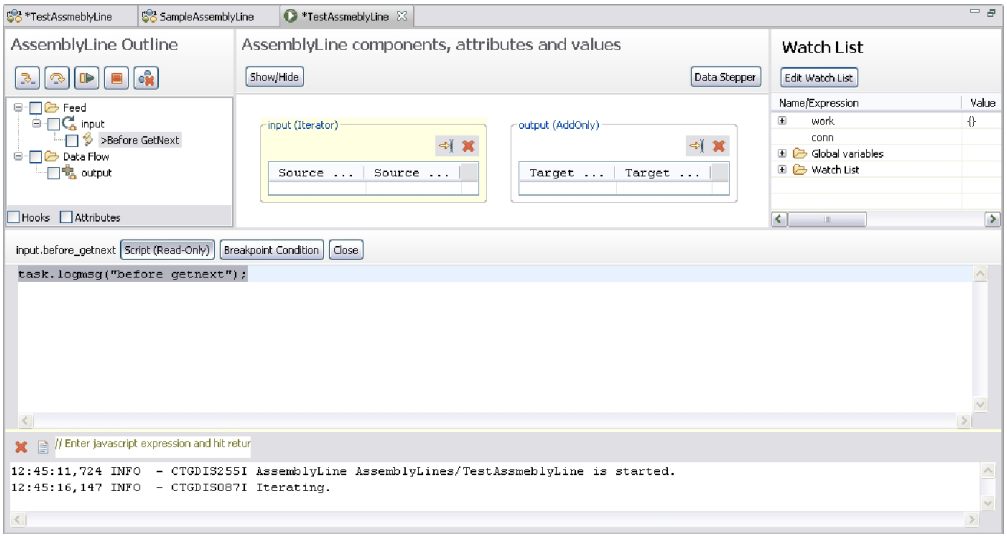


图 89. StepInto 函数

在此 AssemblyLine 中，已在 before init 挂钩中定义了名为 `myfunc1()` 的函数。我们将从 Script1 组件对其进行调用，以显示其出现的方式:

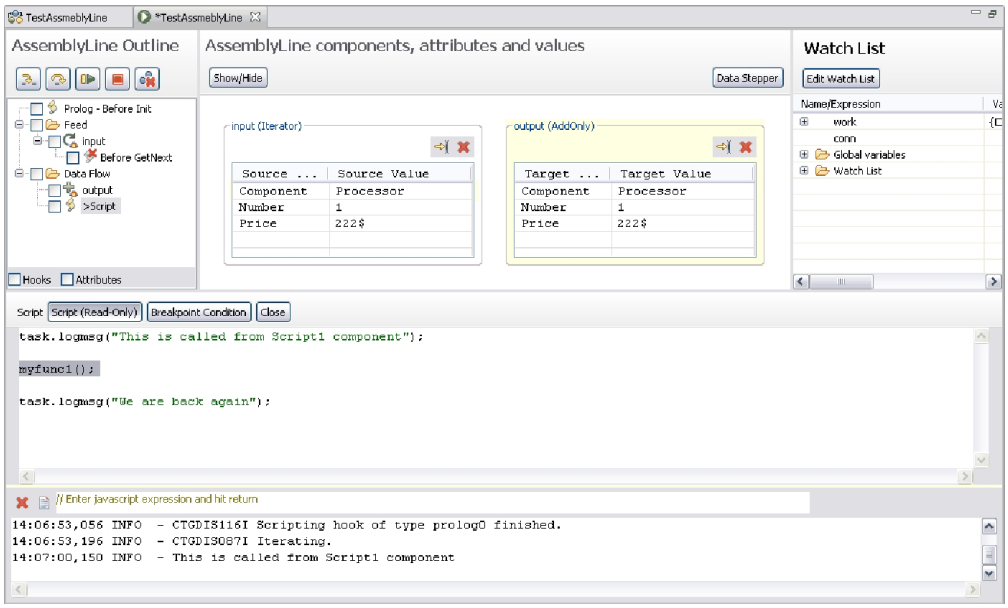


图 90. Stepped-into 函数

此时，可以按 **StepOver** 以继续执行下一条语句，或者按 **StepInto** 以遵循 JavaScript 函数调用:

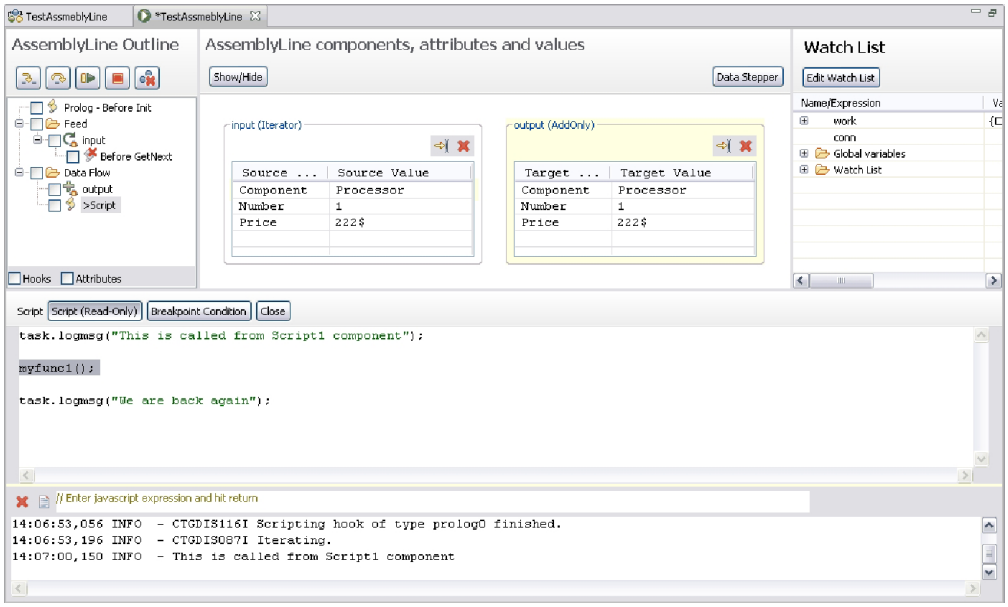
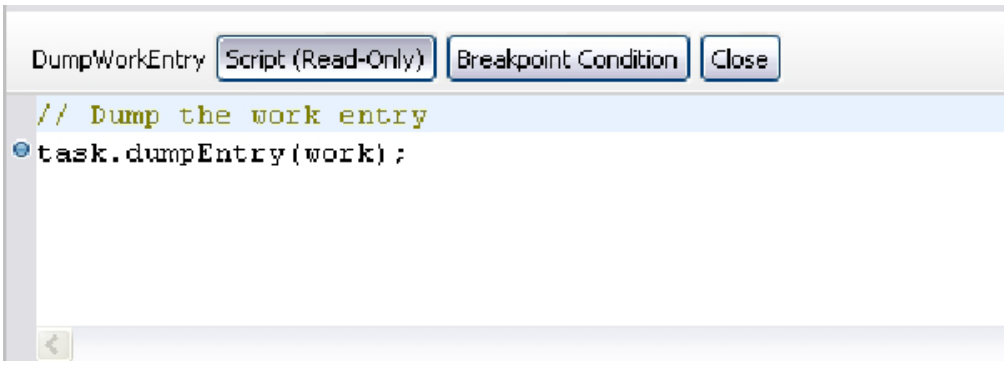


图 91. 遵循函数调用

Step into 导致脚本编辑器从 *before init* 挂钩切换到脚本。请注意用于显示脚本定义位置的已更改标签。

还可以在脚本内设置断点。打开脚本或属性映射的编辑器，并双击左页边距。此时将显示一个蓝色的项目符号，以表示该位置设置了一个断点。再次双击可除去该断点。还可以双击左页边距或文本字段以切换断点。



服务器调试

调试 IBM Security Directory Integrator 服务器意味着在 IBM Security Directory Integrator 服务器上启动的每个 AssemblyLine 会自动与配置编辑器建立调试会话，就像以步进方式启动 AssemblyLine 一样。

通过从“服务器”视图中的服务器上的下拉菜单中选择**调试服务器**，可以激活服务器调试。当选择该选项时，CE 将连接至服务器，并设置指向 CE 的 Java 属性以进行调试。

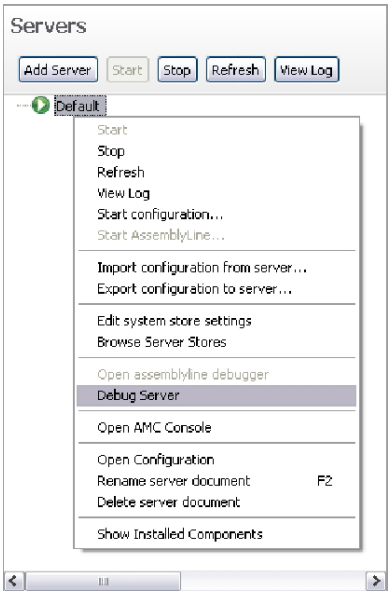
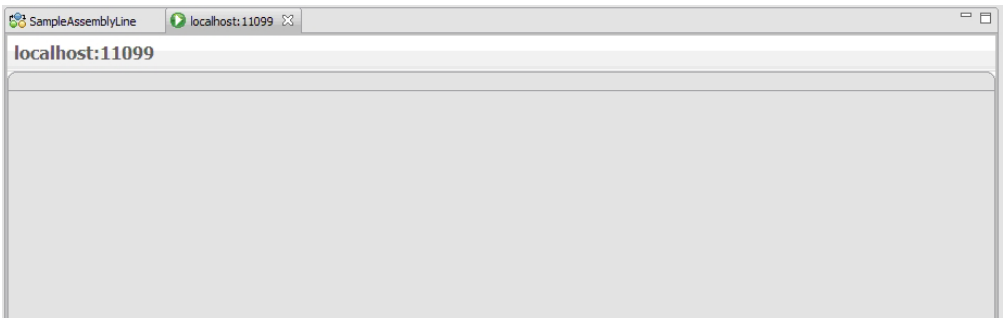


图 92. “调试服务器”选项

选择**调试服务器**将启动新窗口，其中组装流水线将显示为在服务器上启动。这不同于在 CE 中启动 AssemblyLine。从 CE 中启动调试会话时，它将具有其自己的窗口且不会显示在该服务器调试窗口中。



由其他 CE 启动的每个 AssemblyLine 或目标服务器上的组件都在该窗口内具有其自己的步进器面板。请参阅第 139 页的『步进器和调试器』部分，以获取该步进器面板的描述。

运行选项

您可以在运行 AssemblyLine 时指定其他选项。这些选项得到了保存，因此每次运行 AssemblyLine 时都将使用这些选项。

您可以选择运行方式和 AssemblyLine 操作，并为 AssemblyLine 提供初始工作条目（IWE）。

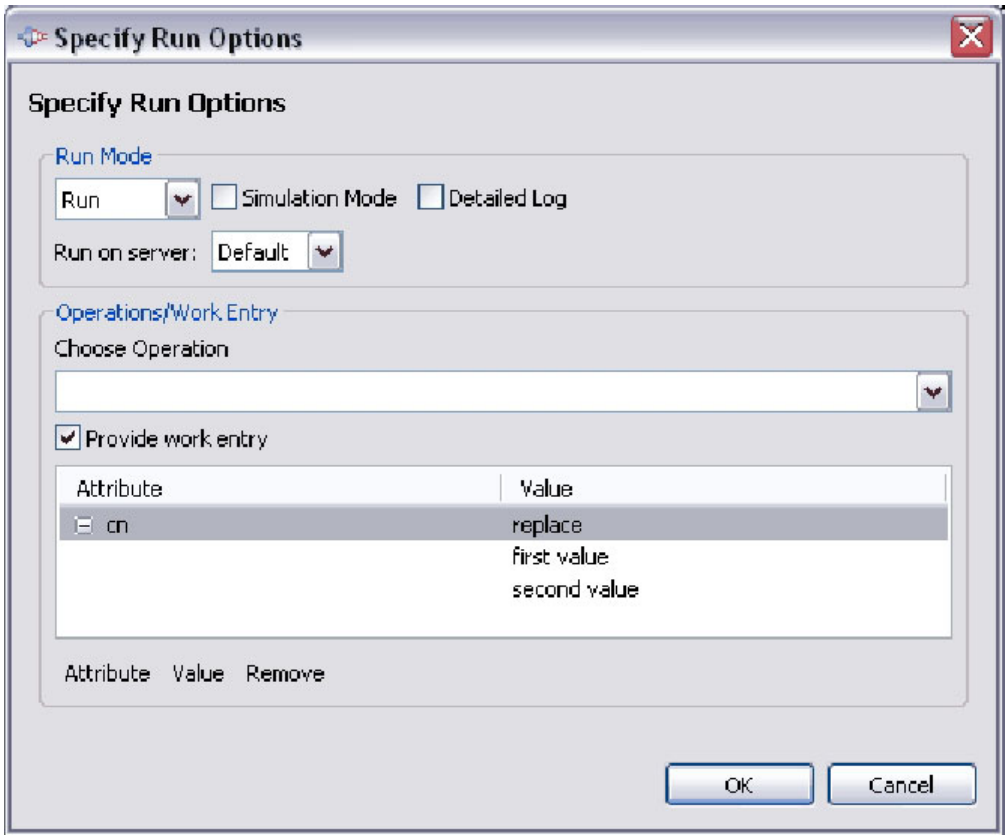


图 93. 带选项运行窗口

使用**属性**和**值**按钮可将属性和值添加到初始工作条目。

选择服务器

运行 AssemblyLine 时，该 AssemblyLine 将在本地开发服务器上执行。该服务器由 CE 启动，且其解决方案目录位于 *TDI 服务器*项目。创建新的 IBM Security Directory Integrator 服务器时，可以通过项目的**属性**对话框更改给定项目的首选服务器:

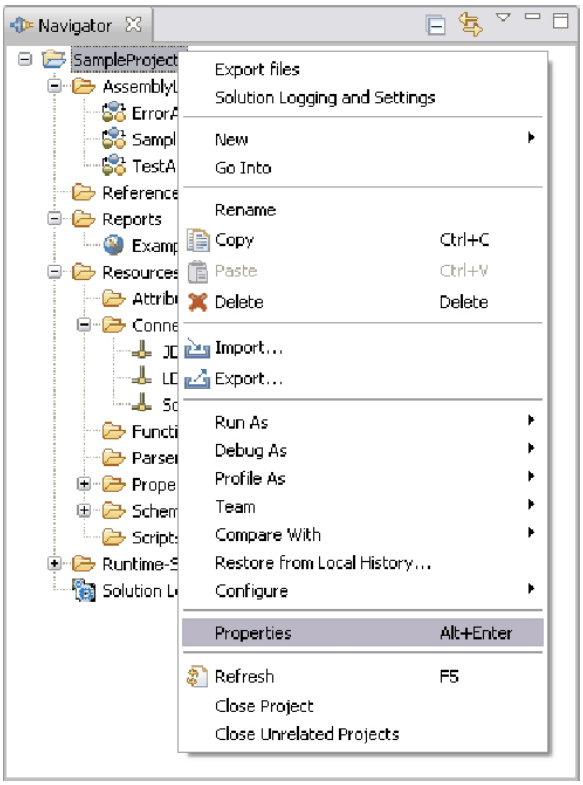


图 94. 项目属性菜单选项

选择该项将启动该项目的属性对话框。选择 **Directory Integrator 属性**以更改项目的缺省服务器。

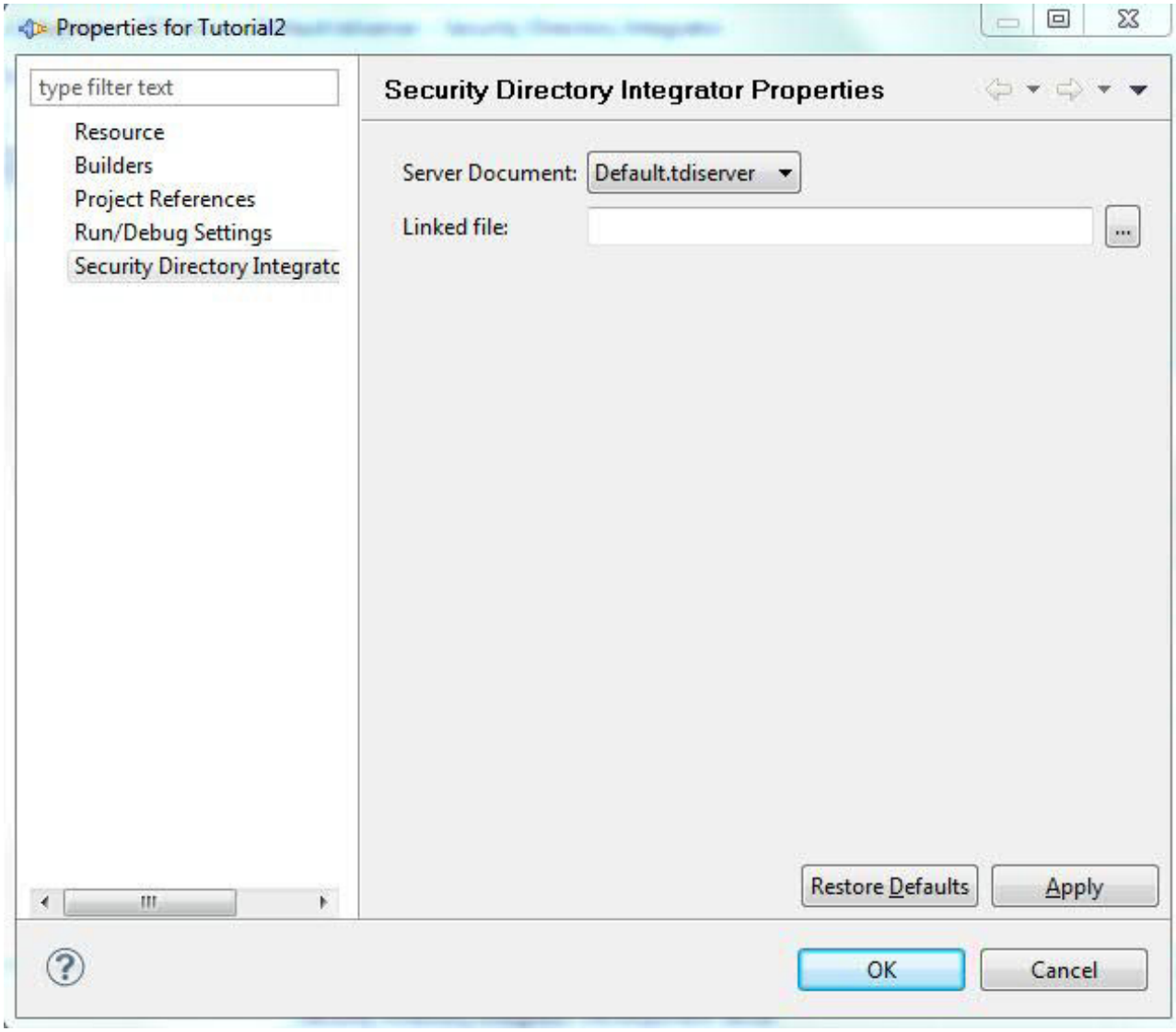


图 95. 项目属性

团队支持

IBM Security Directory Integrator 配置编辑器包含 Eclipse 插件，该插件通过使用源代码控制存储库可在用户之间共享项目。

IBM Security Directory Integrator 包含支持 pserver、pserverssh2、ext 和 extssh 类型连接的 CVS 库。有关 Eclipse CVS 插件的更多详细信息，请访问 Eclipse CVS 站点: http://www.eclipse.org/eclipse/platform-cvs。

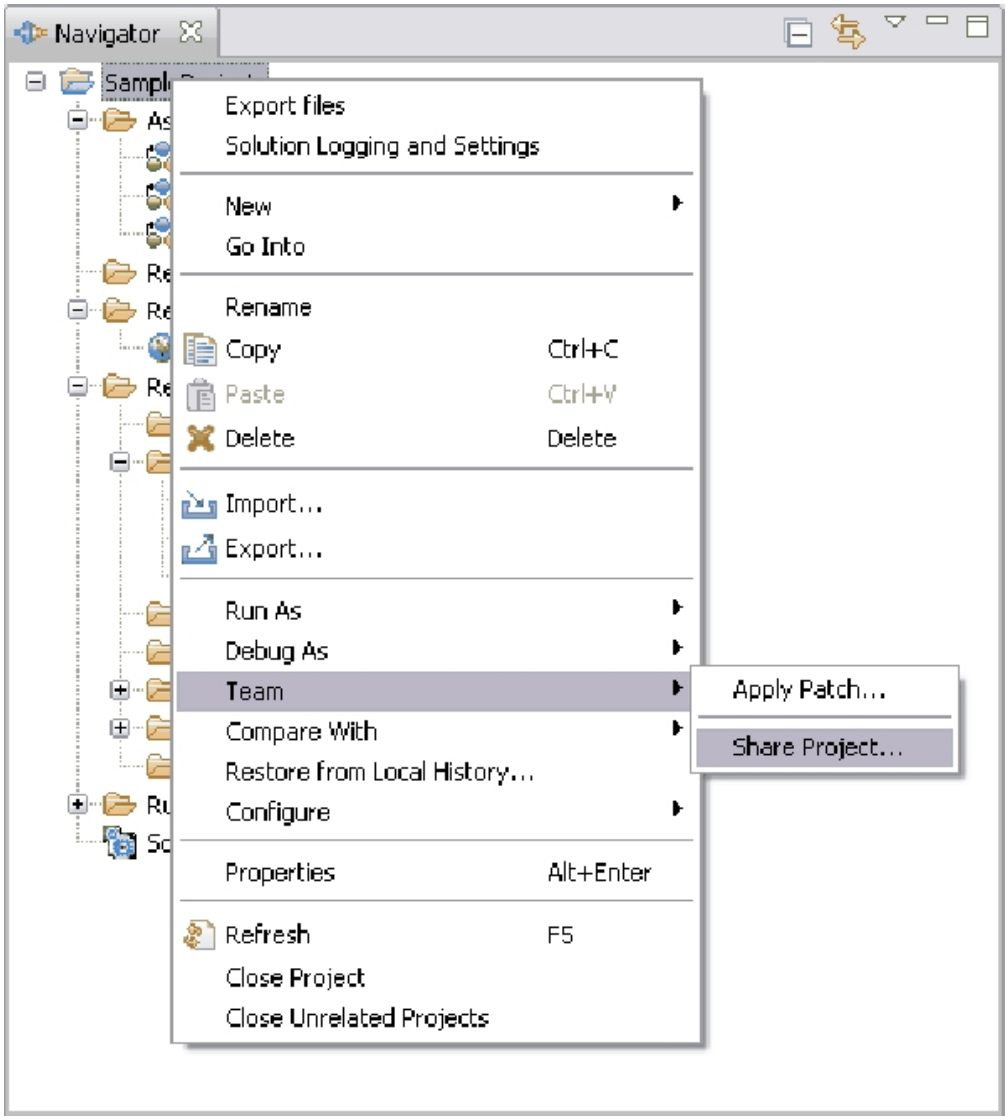
为了使用团队共享设施，需要对 CVS 存储库的访问权。CVS 存储库可以由大多数操作系统主管，且二进制软件包通常可用于该网络。Eclipse CVS 站点具有 FAQ，它有助于您解决可能遇到的大多数常见问题。

有关安装和配置 CVS 服务器的帮助信息，请咨询 CVS wiki 站点：http://ximbiot.com/cvs。在 web 中搜索“how to install a cvs server”将启动 Web 站点的主机，该主机详细描述了如何在各种操作系统和平台上安装和配置 CVS 存储库。

共享项目

您可以使用 CVS 与其他用户共享项目。

要共享项目，请选择项目的下拉菜单中的**团队 > 共享项目...** 选项。



这将打开另一个可以指定源控制存储库的参数的屏幕。

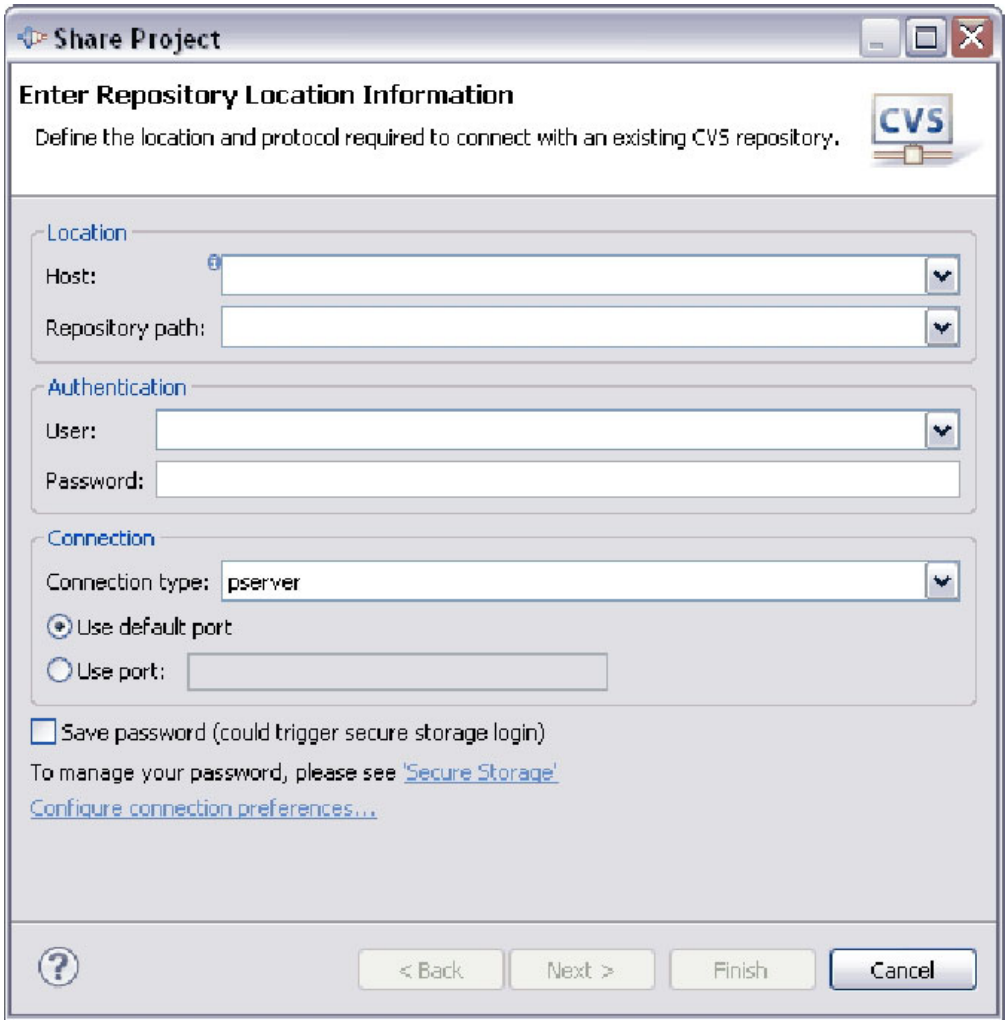
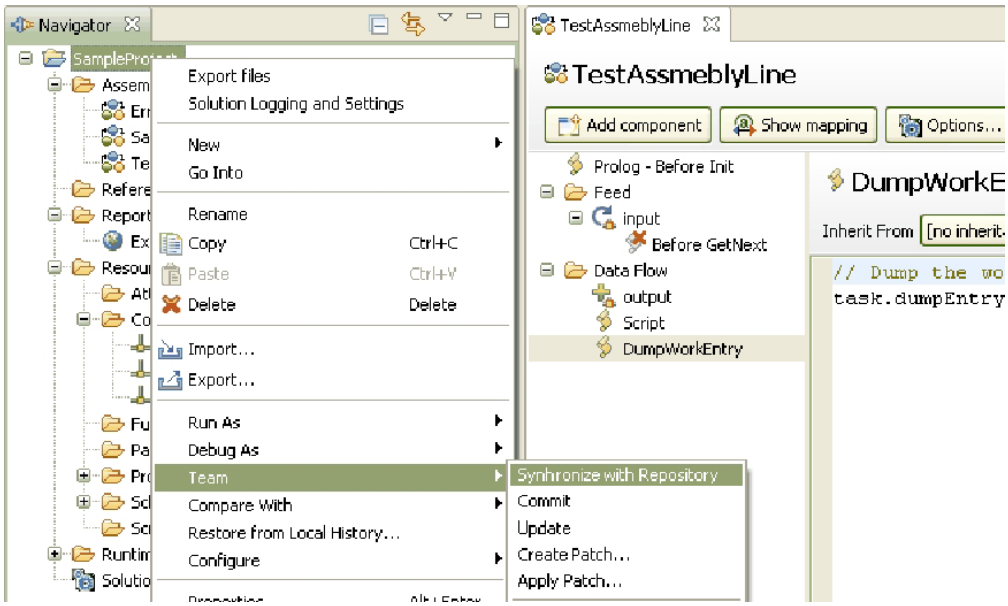


图 96. CVS 共享项目窗口

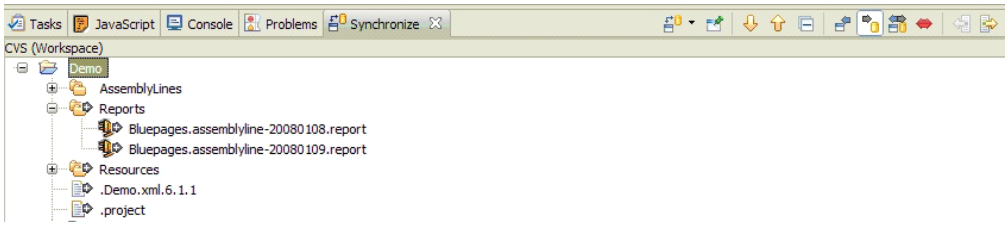
完成向导可将项目共享到 CVS 服务器。

注: 只有项目中的实际文件和包含它们的目录结构可由存储库正确地控制。不考虑空的目录。

在已共享项目后，您可以与存储库同步以落实自己的更改以及接收其他人员作的更改。



选择**团队** > **与存储库同步**打开同步视图:

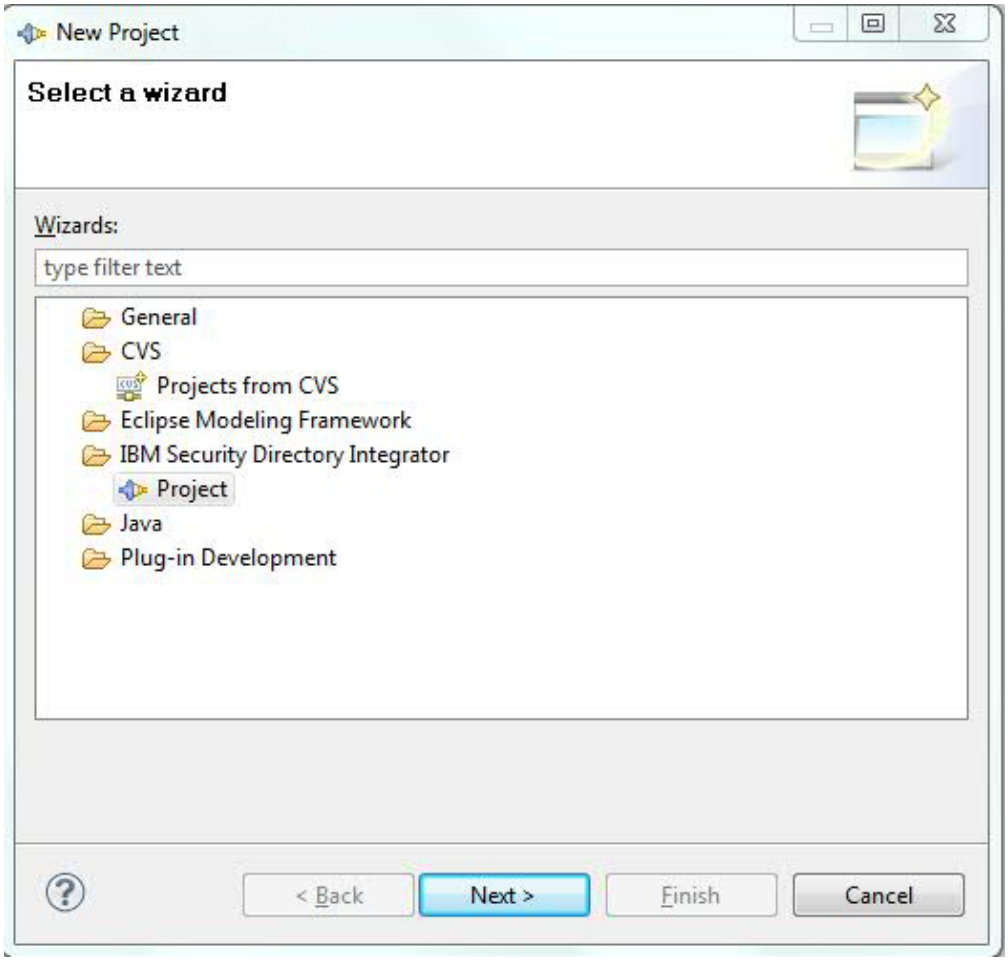


该视图显示需要更新的文件以及有更新的文件。在落实更改后，导航器将显示文件的其他信息。

使用共享项目

如果您希望使用某个人已共享的项目，那么可以使用 CVS 项目向导。

从主菜单中选择**文件** > **新建** > **新建项目...**，然后选择 CVS 项目向导。



完成向导以使项目恢复到工作空间中。

问题视图

当保存组件时，IBM Security Directory Integrator 项目构建器将更新其运行时配置文件以反映已作的更改。

项目构建器还将对组件执行验证检查，并在问题视图中报告警告和错误。在问题视图中，您将看见与以下内容类似的警告:

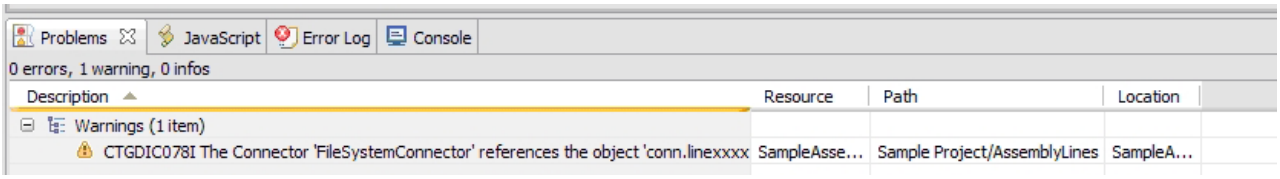


图 97. 问题视图窗口

当双击该视图中的行时，它将打开问题所在的位置。

期望在问题视图中看到的问题包括以下各项:

- 对未定义模式项的引用

如果您使用诸如“conn.abc”的构造，而“abc”在模式中未定义，那么将收到该警告。

- 对未定义工作属性的引用

  如果您使用诸如“work.abc”的构造，而“abc”是否存在未知，那么将收到该警告。
- 脚本中的语法错误
- AssemblyLine 有多个 Server 方式连接器

JavaScript 增强功能

在配置编辑器中您编辑脚本的任何位置，您都会获得一个经过增强而专门用于编辑 JavaScript 代码的文本编辑器。

此编辑器提供代码补全、语法着色以及标记语法错误的功能。

某些增强如下:

- 『代码补全』
- 第 156 页的『语法着色』
- 第 156 页的『语法检查』
- 第 156 页的『本地求值』
- 第 157 页的『外部编辑器』

代码补全

JavaScript 编辑器支持代码补全。

您输入时，编辑器将对某些击键作出反应。此点（.）激活代码补全，该补全会启用弹出菜单，而该菜单显示特定于 IBM Security Directory Integrator 的所有相关方法、字段和功能。还可以通过 `Ctrl-<Space>` 组合键来手动激活代码补全。代码补全涵盖标准 JavaScript 脚本类型（即，字符串、数值等）以及特定于 IBM Security Directory Integrator 的对象的定制补全。诸如 *conn* 和 *work* 之类的对象提供了可用属性的列表，以及对象的字段和方法。

只要编辑器可以在表达式中派生 Java 类名称，代码补全就可运作:

编辑器还会对单引号或双引号以及花括号（“{}”）作出反应。输入单引号或双引号时，编辑器自动插入另一引号并将插入标记置于两者之间。这样做是为了更易于输入字符串常量。

输入花括号将自动缩进以容纳块的缩进。输入左花括号并单击 enter 时，将插入标签，而插入标记以适当的缩进定位在下一行上。相反地，输入右花括号将取消花括号缩进。

映射和代码补全

在 CE 中添加属性时，我们根据属性名称生成简单表达式。包含点的任何名称或不是有效脚本对象标识的其他字符将括在 [″attribute name″] 中（例如: `conn["http.body"]`）。我们使用此表示法时，此条目是否分层已无关紧要，因为其在两种情况下均可行。将按原样使用作为有效 javascript 标识的属性（例如 conn.cn）。如果您有分层模式，那么

我们仍然将为此映射生成简单表达式。例如，如果您拥有 a->b->c 层次结构且您映射“c”部分，那么我们生成 [″a.b.c″] 以引用此属性。

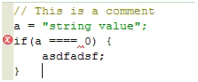
在脚本编辑器中，您将看到代码补全以类似的方式运作。代码补全将以明文显示补全（例如 http.body），但是在单击 Enter 键以完成表达式时，将对并非有效脚本对象名的属性名称使用同一外壳。

语法着色

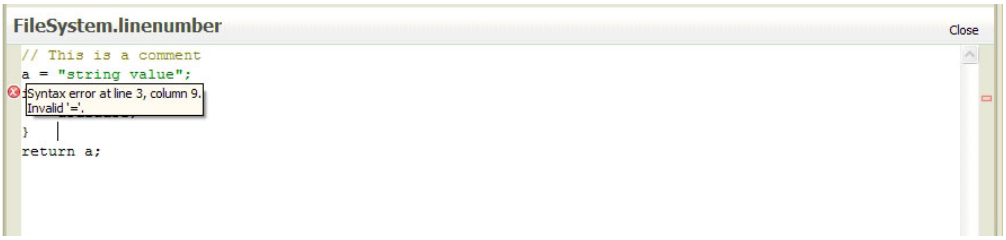
在编辑器中语法着色是基本功能。编辑器会修饰注释和字符串。

语法检查

您输入时，编辑器在后台对脚本执行语法检查。脚本中有错误时，其在边界显示错误，且在文本下划出红波浪线以标记 JavaScript 解释器找到错误的位置。



您将鼠标置于标尺中的错误标记上（左页边距）时，您将在弹出窗口中看到错误消息。

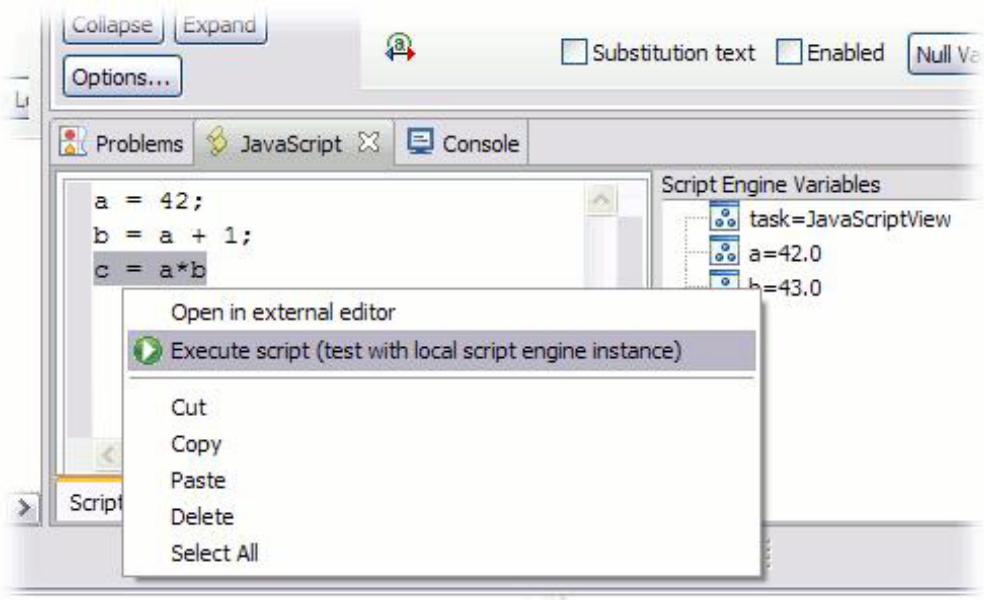


左侧的标尺与文本窗口同步，标记直接与文本窗口中的该行相关。如果您在窗口中的文本内没有看到错误，那么在此标尺内也不会有任何标记。

但是，右侧的概述标尺会显示整个脚本中的所有错误，而不管文本编辑器位于何处。您将鼠标置于标尺中标记之上时，您将获得与在左侧标尺中所获消息相同的弹出错误消息。单击标记会将编辑器重新定位到其中确定问题的行。

本地求值

编辑脚本时，可以右键单击并从下拉菜单中选择**执行脚本**以对选定文本进行快速求值。更详细的脚本测试环境是 JavaScript 视图，可以在其中的单独窗口内执行脚本并查看脚本引擎变量:



外部编辑器

可以使用上下文菜单在您偏好的 JavaScript 编辑器中编辑脚本。

右键单击文本字段并选择**在外部编辑器中打开**。在**窗口** > **首选项** > **Security Directory Integrator** 首选项选项卡中配置该编辑器:

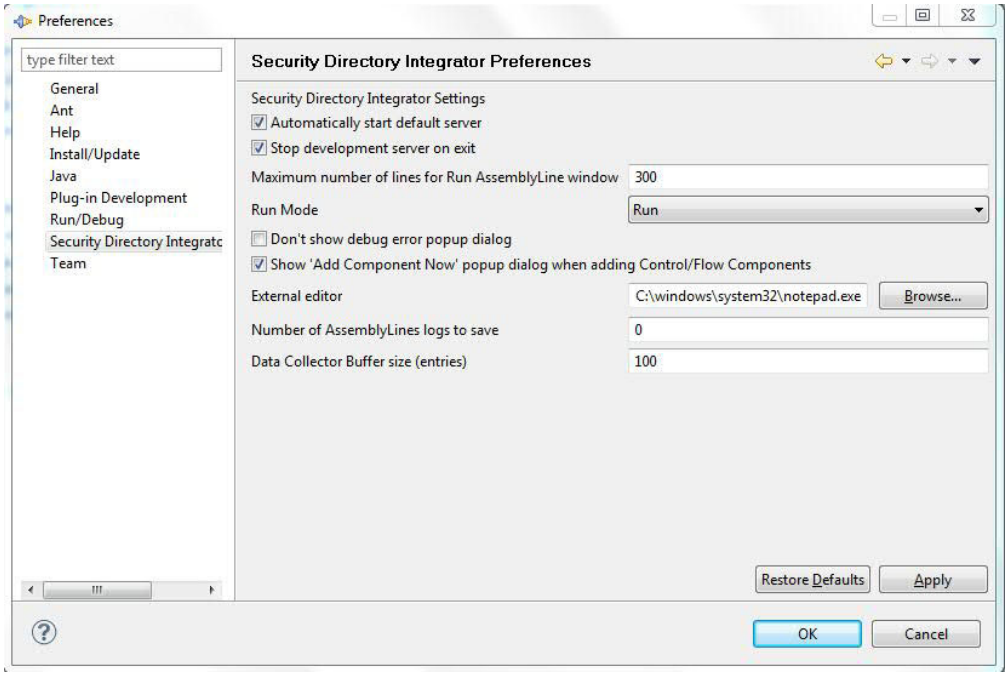
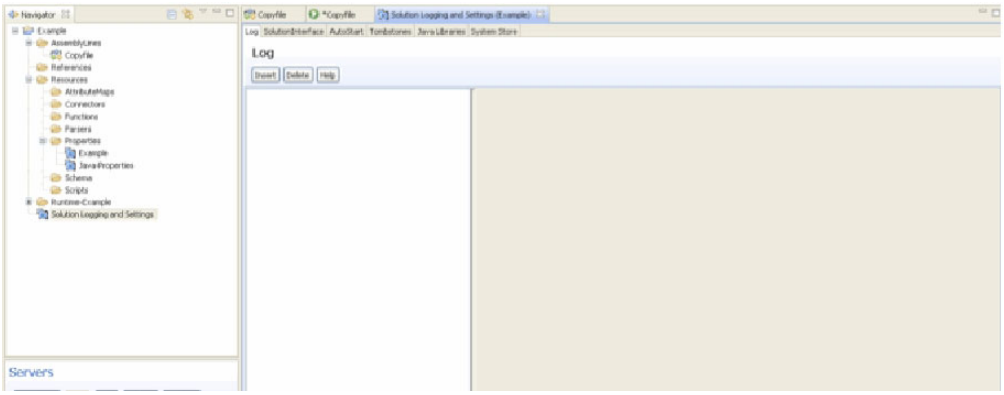


图 98. "配置编辑器首选项"窗口

解决方案记录和设置

通过“解决方案记录和设置”窗口，可以编辑项目的特定于解决方案的区域。

您可以通过选择项目或项目文件来打开此窗口，然后双击项目树中的**解决方案记录和设置**。



在“解决方案记录和设置”下，可以修改下列项:

- 『系统存储设置』
- 第 160 页的『记录日志』
- 第 160 页的『墓碑』
- 第 161 页的『Java 库』
- 第 161 页的『自动启动』
- 第 161 页的『解决方案接口设置』

系统存储设置

项目的系统存储设置可以覆盖由 IBM Security Directory Integrator 服务器定义的缺省系统存储。在启用了此设置时，配置将使用已配置的系统存储而不是服务器的系统存储。

可以在两个位置进行系统存储设置:

1. 在工作空间中的“IBM Security Directory Integrator 项目”级别；以及
2. 在“IBM Security Directory Integrator 服务器”级别。

项目级别的设置优先于服务器级别的设置，也即是说，如果已为项目专门定义了系统存储，那么在配置编辑器中运行组件时将使用该系统存储；如果尚未在项目中定义任何系统存储设置，那么将使用用于运行组件的服务器的系统存储。

项目级设置

在标题的下拉菜单中，您可以选择预定义的模板以及将系统存储设置装入并保存到工作空间中的文件。请注意，所有表（变化量和属性等）都将存储在该数据库中。

列为**已嵌入 Derby** 等等的菜单项是可以装入到配置面板中的预定义模板，此后可以根据需要将其修改，然后在配置中对其进行更新。此外，也可以从本地文件系统**装入模板...**。

对此面板进行的更改会保存在配置文件中（当导出了项目时），并且供其所有 AssemblyLine 使用，而与执行这些更改所在的服务器的设置无关。

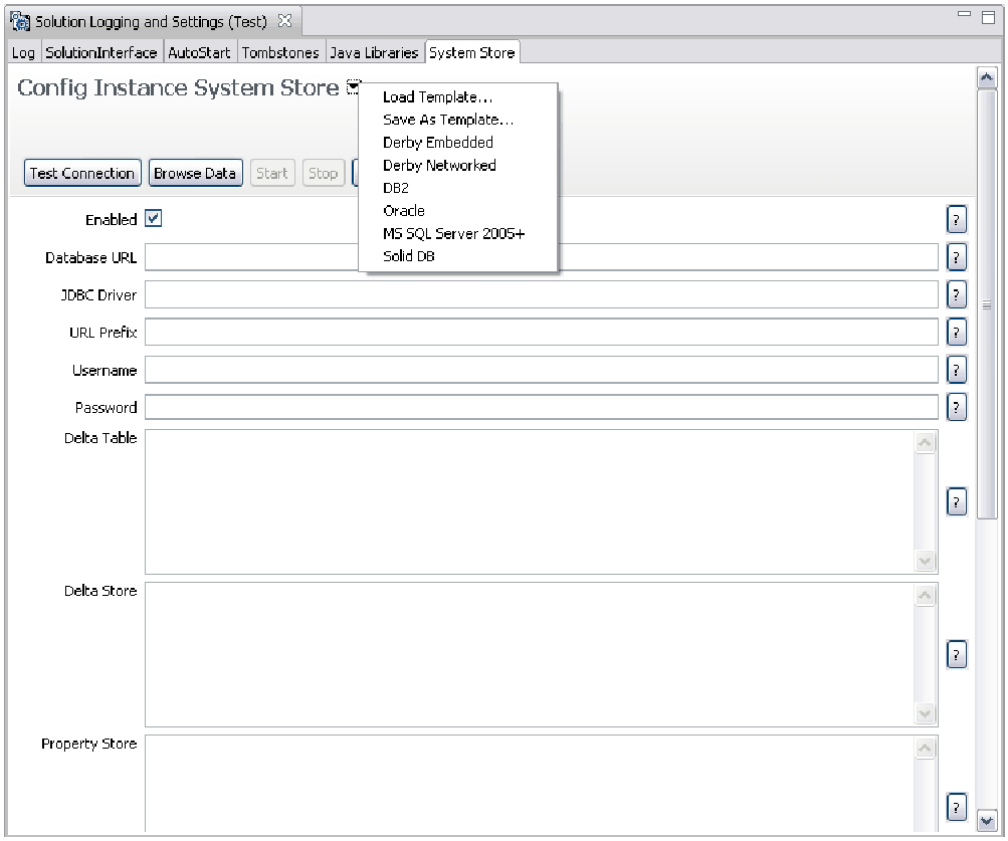


图 99. 配置系统存储设置

服务器级设置

当从“服务器”视图内服务器的下拉菜单中选择**编辑系统存储设置**时，将出现相同的屏幕。这将更新系统存储设置变量，并将其保存到解决方案属性文件中。需要重新启动服务器才能激活新设置。

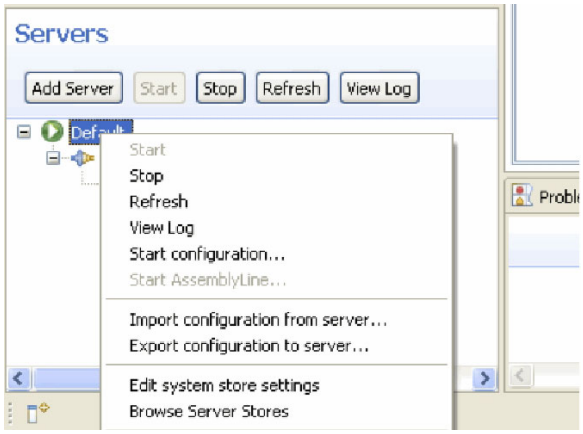
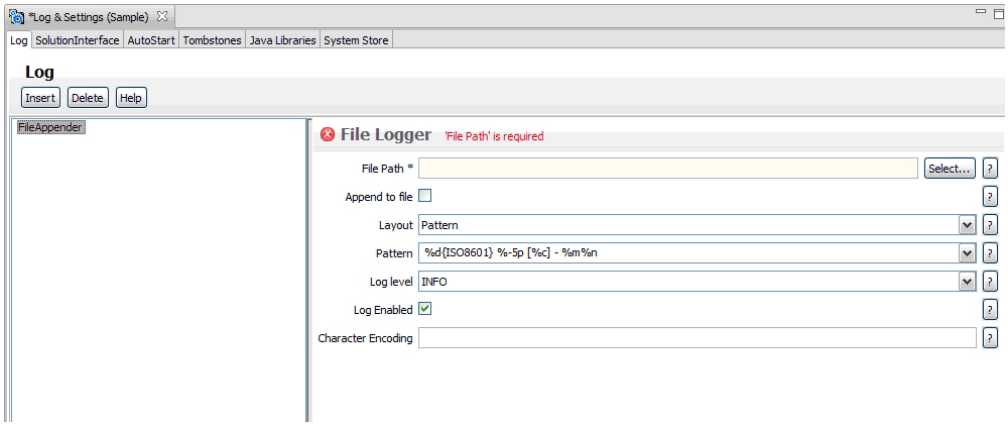


图 100. 服务器文档上下文菜单

记录日志

记录视图显示了解决方案的记录器。

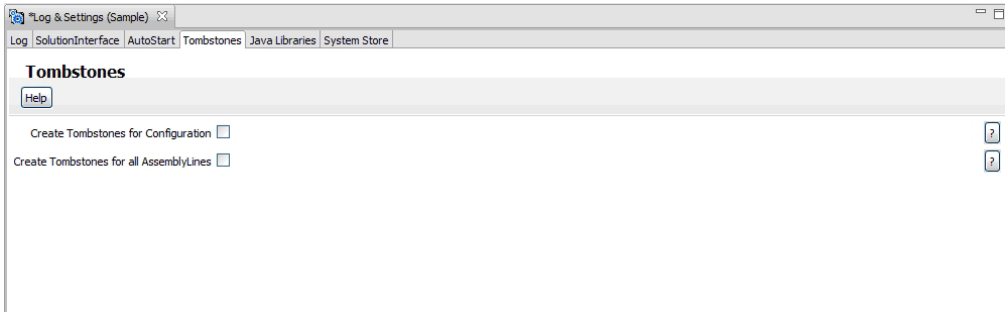
您可以通过单击标题栏中的**插入**和**删除**来添加和除去记录器。



请参阅*安装与管理*中的“记录和调试”部分，以获取更多信息。

墓碑

在**墓碑**选项卡中可以找到项目的墓碑配置。



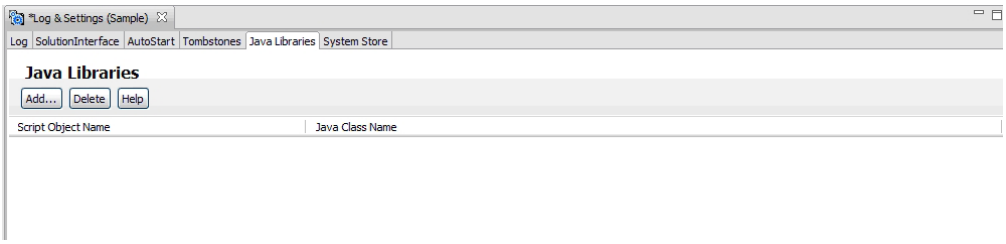
墓碑是“AssemblyLine”运行的记录，包括一些统计信息（例如迭代器读取的条目数、已跳过的记录数和已更新的记录数等等）。

选择**创建配置的墓碑**会使源于该项目（在 IBM Security Directory Integrator 服务器上装入）的配置文件每次终止时都生成墓碑。

选择**为所有 AssemblyLine 创建墓碑**在该项目中的“AssemblyLine”（在服务器上运行）每次终止时都将生成墓碑。

Java 库

Java 库选项卡显示了每个脚本引擎实例中自动装入和定义的 Java 类。



自动启动

自动启动选项卡显示了可以输入 AssemblyLine（在启动配置实例时自动启动）的名称的列表。

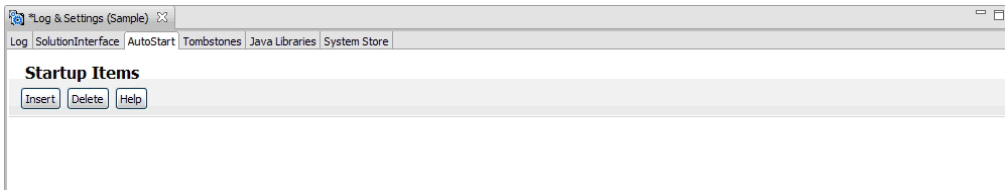


图 101. 自动启动设置

您可以通过单击**插入**将项添加到列表；这使您能够从工作空间中的现有 AssemblyLine 进行选择。

您可以通过选择 AssemblyLine 并单击**删除**从“启动项”列表中除去 AssemblyLine。

解决方案接口设置

启用解决方案接口设置可提供其他关于此配置的信息。该信息通常由 Web 管理工具（AMC）使用，但也可由其他能访问解决方案接口配置的客户机使用。

该面板中的前两个字段是解决方案名称和已启用状态。解决方案名称的缺省值是项目名称自身；如果解决方案名称保留为空，那么已导出的配置文件将不包含任何解决方案标识。已启用的复选框确定配置是否正在使用中。

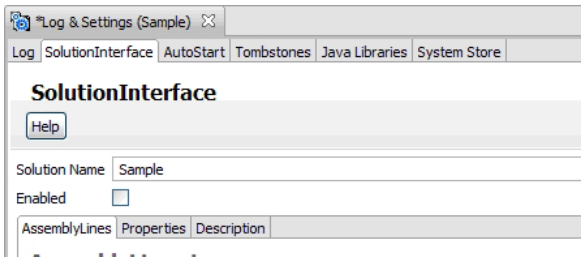


图 102. 解决方案接口设置

解决方案接口设置分为三个部分，每个部分都有相应的选项卡:

- 第 162 页的『AssemblyLine』

- 『属性』
- 第 163 页的『描述』

AssemblyLine

该选项卡显示项目中的所有 AssemblyLine，并可以选中那些应该对用户可见的 AssemblyLine 以执行启动/停止操作。

运行状况 AssemblyLine 是一种特殊的 AssemblyLine，用于报告正在运行的配置的运行状况。该 AssemblyLine 仅可以向用户提供关于配置状态的定制反馈。将调用运行状况 AssemblyLine，并应返回其工作条目中的两个字段以报告状态（`healthAL.result` 和 `healthAL.status`）。请参阅操作管理器（AM）文档（联机时在 AMC 中或在 *安装与管理* 中）以获取更多关于如何在该上下文中使用这些字段的信息。轮询时间间隔指定客户机（例如 AMC 或 AM）将调用运行状况 AssemblyLine 的频率。

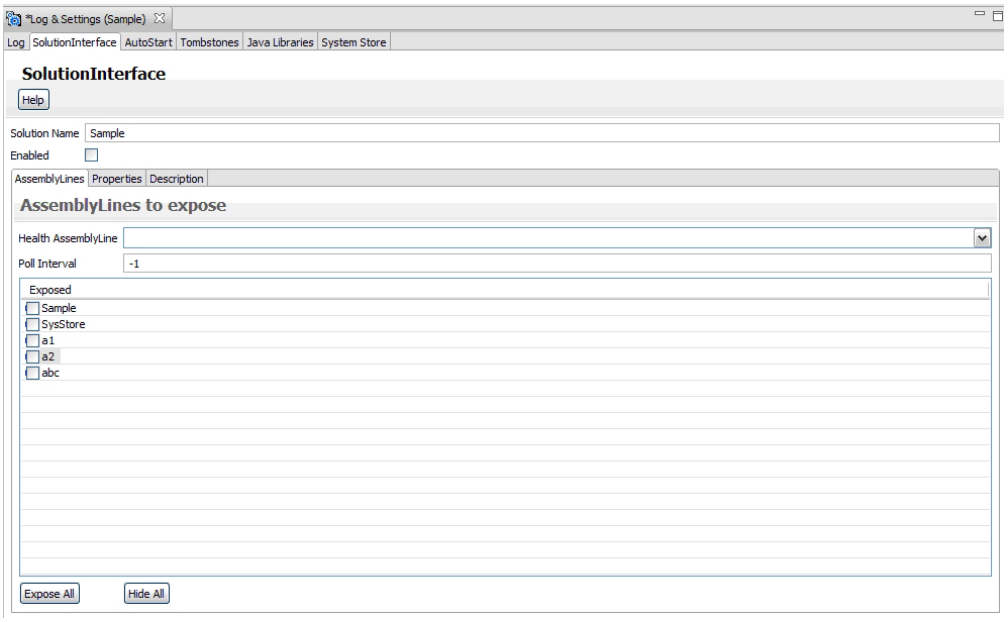


图 103. 解决方案接口设置：AssemblyLine

属性

该选项卡使您能够定义用户可以看见的属性以及向用户显示属性的方式。

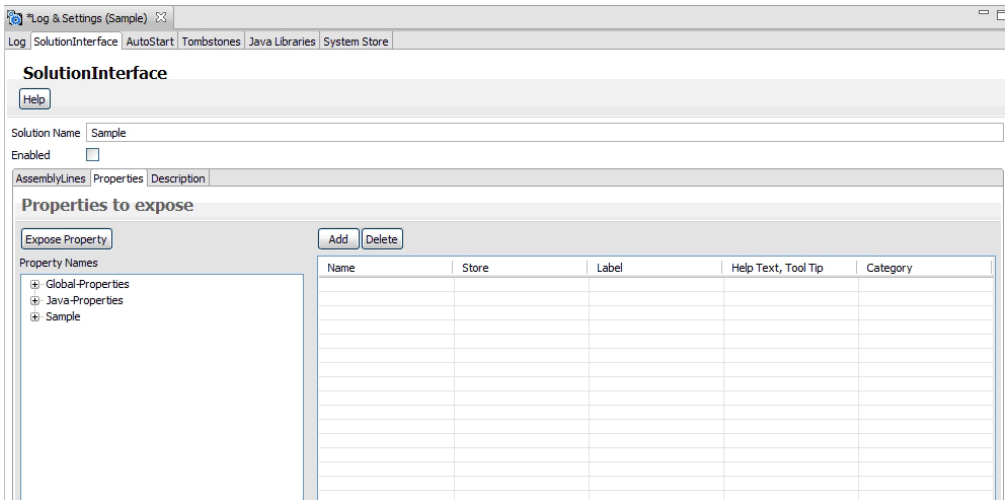


图 104. 解决方案接口设置: 属性

描述

该选项卡使您能够输入描述解决方案的文本。它仅用于文档目的; IBM Security Directory Integrator 不会以其他任何方法使用该文本。

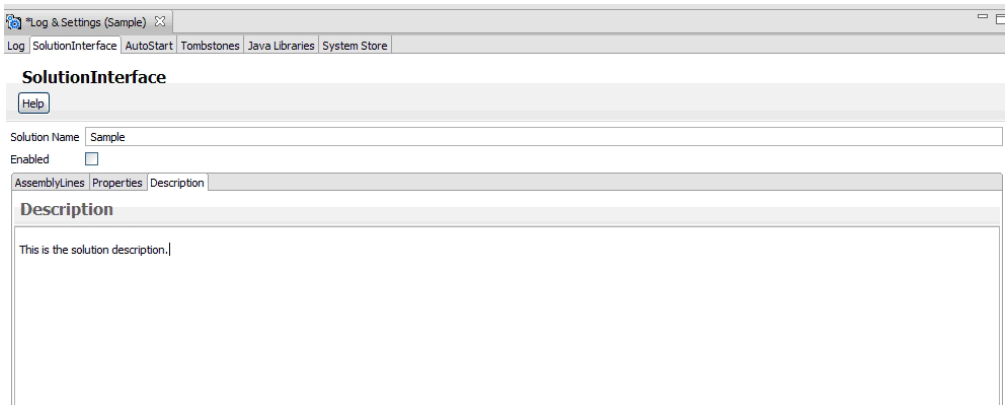


图 105. 解决方案接口设置: 描述

服务器属性

您可以使用“属性”编辑器来编辑项目的属性。

使用**文件 > 新建 > 属性**向导创建新的属性文件并输入属性存储的名称。

在属性编辑器中，可以使用**下载**和**上载**命令交换属性存储的内容:

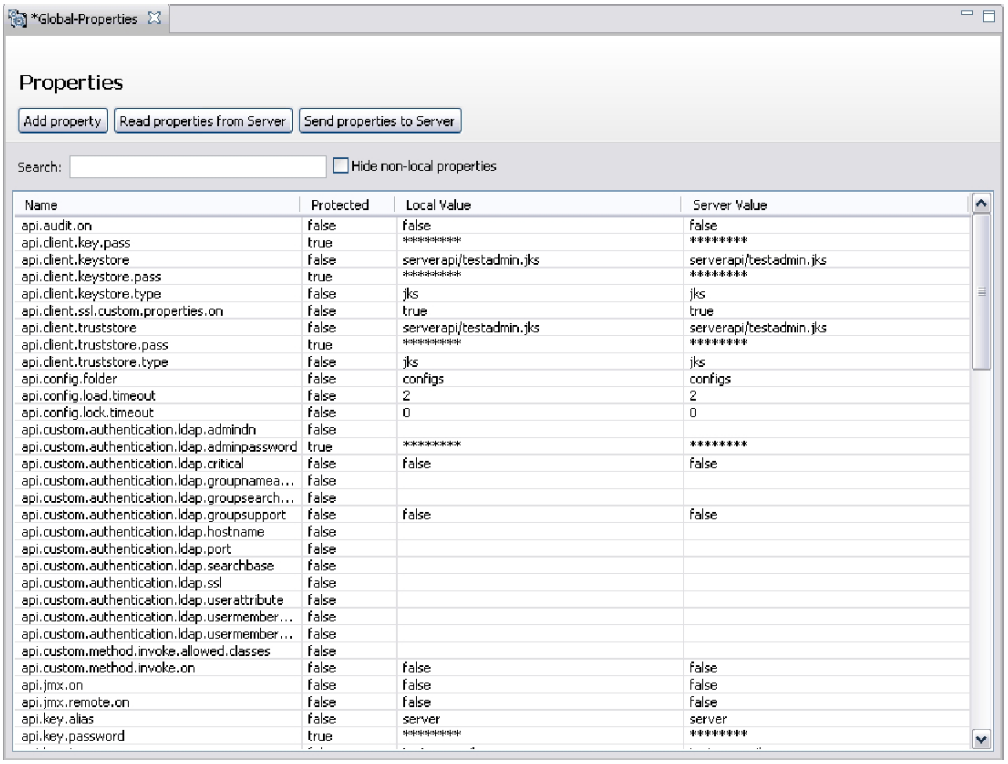


图 106. 属性编辑器窗口

使用**下载**可以检索属性存储器中的所有属性（例如，指定给此存储器的属性文件）。请注意，这些值是从当前为项目选择的 IBM Security Directory Integrator 服务器读取并传递到 CE 的。反之，**上载**按钮将使用编辑器中的值来更新属性存储器（通过当前服务器）。只更新那些有本地值的属性。

使用**搜索**文本字段可显示与该字段中的文本匹配的属性。选中**隐藏非本地属性**导致编辑器仅显示具有本地值的属性。

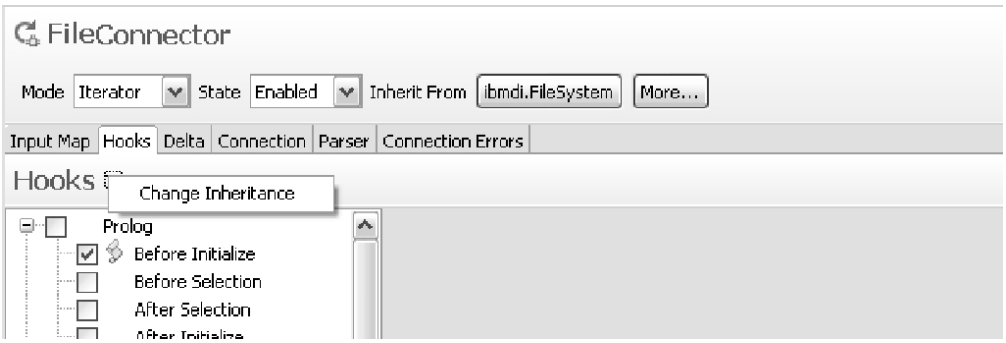
项目构建器将包含可运行配置文件中的属性存储配置。但是，该文档中的属性值仅按需要传送。

注: 在 IBM Security Directory Integrator 服务器中初始化和访问属性存储的顺序未定义。因此，不可能在属性存储中可靠地存储定义其他属性存储的访问参数（例如文件名）的任何属性。

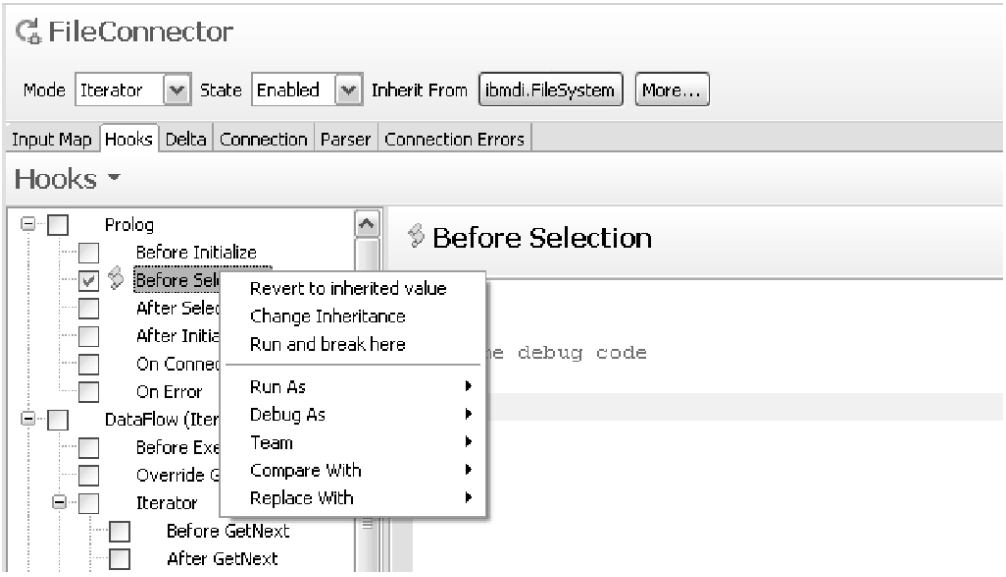
继承

组件可以通过将对象拖放到组件的标题栏或组件的子节来从其他组件继承元素。

对于子节，在标题栏上还有可用的菜单选项（**更改继承**）。



对于挂钩和属性映射，在各个项上有单独的继承选项，您可以在其中选择从工作空间中的脚本或函数组件进行继承。

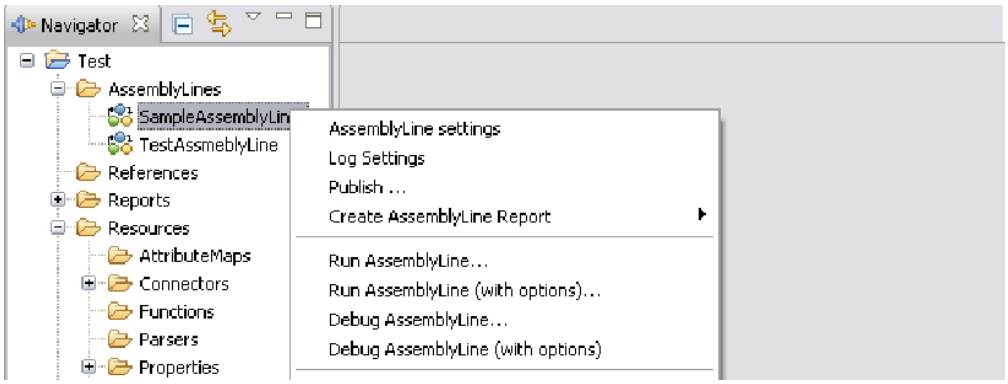


配置项覆盖继承项时，用户界面提供一种方式来通过下拉菜单操作项（**还原到继承的值**）来还原到继承的值。

操作和键绑定

CE 为其自身以及工作台上的对象提供了大量操作。这些操作对特定对象执行特定操作。

例如，**运行 AssemblyLine 报告**是提供给扩展名为 `.assemblyline` 的所有文件的操作。当您右键单击导航器上的 AssemblyLine 时，下拉菜单将包含该命令以及提供给该类型对象的所有其他命令。



这些操作还具有关联的命令定义。命令定义可使用户定义命令的键盘快捷键。该操作在 **Window** > **首选项** > **键**面板中执行。

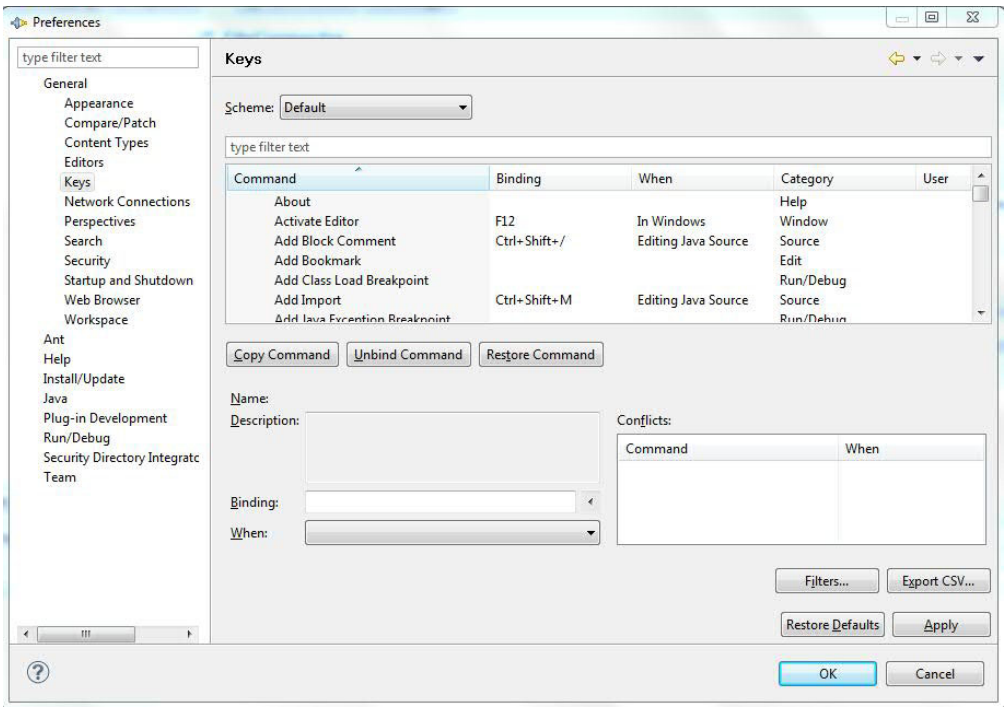
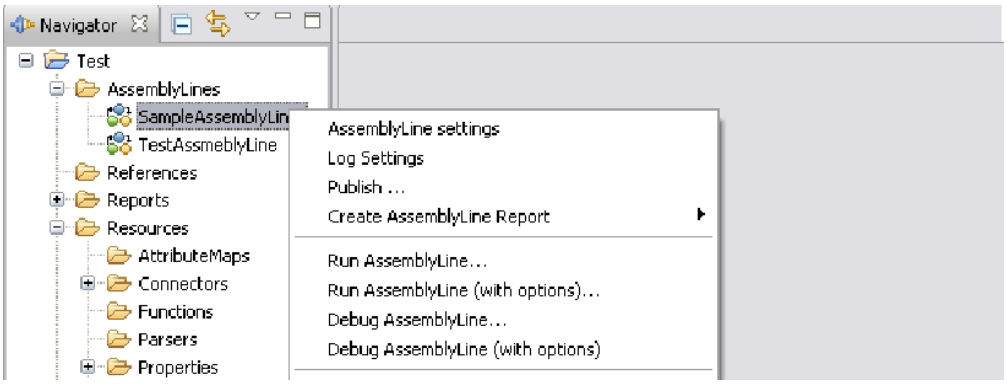


图 107. “键指定”窗口

以上图片显示如何在用户界面中执行键指定。在该示例中，运行报告命令已将 `Alt+Shift+I` 指定为其快捷键。当您在组装流水线上再次打开菜单时，将看到反映在菜单中的该快捷键。



在搜索字段中输入 `security directory integrator` 文本后可获取所有特定 IBM Security Directory Integrator 命令的列表，该列表位于**方案**选择器的下面。

第 4 章 IBM Security Directory Integrator 中的调试功能

一旦创建了 IBM Security Directory Integrator 解决方案（通常使用配置编辑器（CE）），有多种方法可以完成该解决方案并对其进行测试。IBM Security Directory Integrator 提供的部分调试功能部件可从 CE 中获取，而另一些功能部件则取决于 IBM Security Directory Integrator 安装目录中提供的脚本。

IBM Security Directory Integrator 中的调试功能部件包括沙箱、AssemblyLine 模拟方式以及步进器和调试器。

步进器和调试器都是配置编辑器的一部分，请参阅第 139 页的『步进器和调试器』。

沙箱

IBM Security Directory Integrator 包含沙箱功能部件，该功能部件使您能够记录 AssemblyLine 中一个或多个连接器的操作，以便稍后重放而无需提供必要的数据源。

“沙箱”功能部件使用系统存储器。关于更多信息，请参阅第 183 页的第 6 章，『系统存储』。

记录组件意味着 AssemblyLine 将拦截对组件使用的连接器实例的每次调用。例如，记录连接器组件将记录所有对该连接器实例方法（例如 `selectEntries` 和 `getNextEntry` 等等）的调用。每个这些调用的结果都记录在已配置的沙箱数据库中。记录的信息是返回值或由连接器抛出的异常。

要运行测试会话，请创建 AssemblyLine 并添加从文件读取数据的文件系统连接器。添加转储工作条目的脚本组件。然后检查该面板中的文件系统连接器，并选择“运行”按钮菜单上的**运行/记录**。在完成此步骤之后，您可以移动刚读的文件，并用“回放”方式运行 AssemblyLine。尽管文件连接器会由于文件不再可访问而正常终止，但应该可看见相同的日志输出。

提供支持资料时，该功能部件会十分有用。通常，重新生成 AssemblyLine 的环境和数据源的状态以重新生成条件的时间可能十分漫长。使用带有记录的会话的沙箱数据库，支持人员可以运行 AssemblyLine 而不用访问该 AssemblyLine 需要的所有数据存储。除此之外，可以修改 AssemblyLine 配置以打印出更多的信息（如果需要这么做的话）。唯一不能对 AssemblyLine 配置作的更改是进行其他调用或对已记录组件的调用重新排序。这将导致回放期间出现错误，因为对连接器的调用将不会与下一个预期连接器调用匹配。

在可以记录或重放 AssemblyLine 之前，必须先告知 IBM Security Directory Integrator 在何处存储记录数据的 AssemblyLine。该操作在“沙箱”窗口中完成，通过在 AssemblyLine 编辑器中选择 **AssemblyLine 设置... > 沙箱设置**可打开该窗口。该窗口的顶部是一个标有**数据库**的字段，您可以在其中输入系统使用的目录路径。

在包含处于 Server 方式的连接器或者包含启用了 Delta 的 Iterator 连接器的 AssemblyLine 中不支持“沙箱”工具。如果发现此情况，服务器将异常终止 AssemblyLine 的运行。

记录 AssemblyLine 输入

记录组件意味着 AssemblyLine 将拦截对组件使用的连接器实例的每次调用。例如，记录连接器组件将记录所有对该连接器实例方法（例如 `selectEntries` 和 `getNextEntry` 等等）的调用。每个这些调用的结果记录在已配置的“沙箱”数据库中。记录的信息是返回值或由连接器抛出的异常。

要运行测试会话，请创建 AssemblyLine 并添加从文件读取数据的文件系统连接器。添加转储工作条目的脚本组件。然后检查该面板中的文件系统连接器，并选择**运行**按钮菜单上的**运行/记录**。在完成此步骤之后，您可以移动刚读的文件，并用“回放”方式运行 AssemblyLine。尽管文件连接器会由于文件不再可访问而正常终止，但应该可看见相同的日志输出。

AssemblyLine 记录的沙箱回放

当一个 AssemblyLine 处于 Sandbox 方式时，所有设置回放的连接器都处于*虚拟方式*。这意味着实际上不调用它们的连接器接口操作（例如，`getNext()` 和 `findEntry()`）。回放的过程中会模拟这些操作。

为了在沙箱回放方式下运行 AssemblyLine，必须通过 AssemblyLine **沙箱**设置窗口中对应的**回放已启用**复选框来选择将要在虚拟方式下运行的连接器。

注: 并非所有记录的连接器都需要启用以进行回放。可以启用它们来访问活动的数据源，尽管这可能影响回放操作的结果。

要从命令行运行 AssemblyLine，请带 **-q2** 开关启动服务器。处于沙箱方式的 AssemblyLine 运行时带有来自已记录数据集的输入（包括“初始工作条目”）。例如，如果您在处于沙箱方式的 AssemblyLine 中具有“Java 消息服务 (JMS)”连接器，那么 JMS 连接器从先前记录的数据中检索输入并且从不进行实际的初始化。

记录 AssemblyLine 时，服务器会使用 AssemblyLine 名称作为数据库名称在指定的**数据库**目录中创建 Derby 数据库。该数据库包括了用于 AssemblyLine 中的每个连接器的表。处于沙箱方式的 AssemblyLine 可以通过重命名已记录的连接器，然后添加一个带有其初始名称的新连接器，来替换其中的一个或多个虚拟连接器。

AssemblyLine 模拟方式

可以调试 AssemblyLine 而不需要使用 AssemblyLine 模拟方式来与已连接的系统实际交换数据。当以该方式启动 AssemblyLine 时，将跳过所有可能导致目标系统中的更改的 AssemblyLine 组件。

这意味着 AL 执行正常，但处于 Update、Delta、Delete 和 AddOnly 方式的连接器不执行实际操作。

注: 以此方式运行的 AssemblyLine 只能大致模拟正常方式会发生的情况；在许多情况下，已连接系统不会接收到任何模拟方式更新，这可能导致业务逻辑表现出不同的行为，从而抵消模拟的有用性。

不应该将模拟状态与组件的状态混淆。组件的状态比组件的模拟状态的优先级更高。

组件的状态有两个值 –“已启用”或“已禁用”。如果组件处于“已启用”状态，那么它将进行初始化，并且在调用相应的操作时检查其模拟状态。如果将状态设置为“已禁用”，那么将不会对模拟状态作任何检查，因为不会调用任何操作，并且不会调用组件的初始化。

连接器和 FC 有另一个称为“被动”的状态。如果将连接器和 FC 的状态设置为“被动”，那么仅会对它们执行初始化。只有通过用户脚本才可以执行它们的操作。如果调用它们的特定操作，那么将检查模拟状态。

有几种方法可以用模拟方式启动 AssemblyLine:

从配置编辑器:

运行方式下拉菜单中的复选框可用于启用和禁用 AL 的模拟。您必须从下拉菜单中选择想要的运行方式“带选项运行”，并选中弹出的“选项”对话框中的复选框**模拟方式**以使 AL 在运行时执行模拟。该复选框的缺省状态是未选中的。

使用服务器启动命令 `ibmdisrv`:

如果提供该 switch 语句，那么该命令可识别 switch **`–M`**，并且将启动已打开模拟的 AL。

使用 API:

为了使 AL 在模拟方式下运行，您必须在 TCB 对象中将属性 `AssemblyLine.TCB_SIMULATE_MODE` 设置为 true。然后应该将该对象提供给 `startAssemblyLine(String, TaskCallBlock)` 方法。如果未设置任何属性，那么缺省情况下认为其值为 false。

为了以模拟方式运行 AssemblyLine，将创建一个新的配置。它是 AssemblyLine 配置的子代。该配置对象中包含的某些参数可用于配置与 AL 的连接，此 AL 将用作代理 AssemblyLine (ProxyAL)。SimulationConfig 对象具有一个方法，用于根据要模拟的 AssemblyLine 的组件状态，创建或更新 ProxyAL 的模板。SimulateConfig 对象还包含为处于“已编制脚本”模拟状态的组件定义的所有挂钩。每个挂钩的名称与处于“已编制脚本”模拟状态的组件的名称相同。它还包含每个组件的模拟状态。

可以更详细地配置将在模拟方式下运行的 AssemblyLine；通过在“AssemblyLine 编辑器”窗口中选择**AssemblyLine 设置 > 模拟设置**可以配置相关的设置。

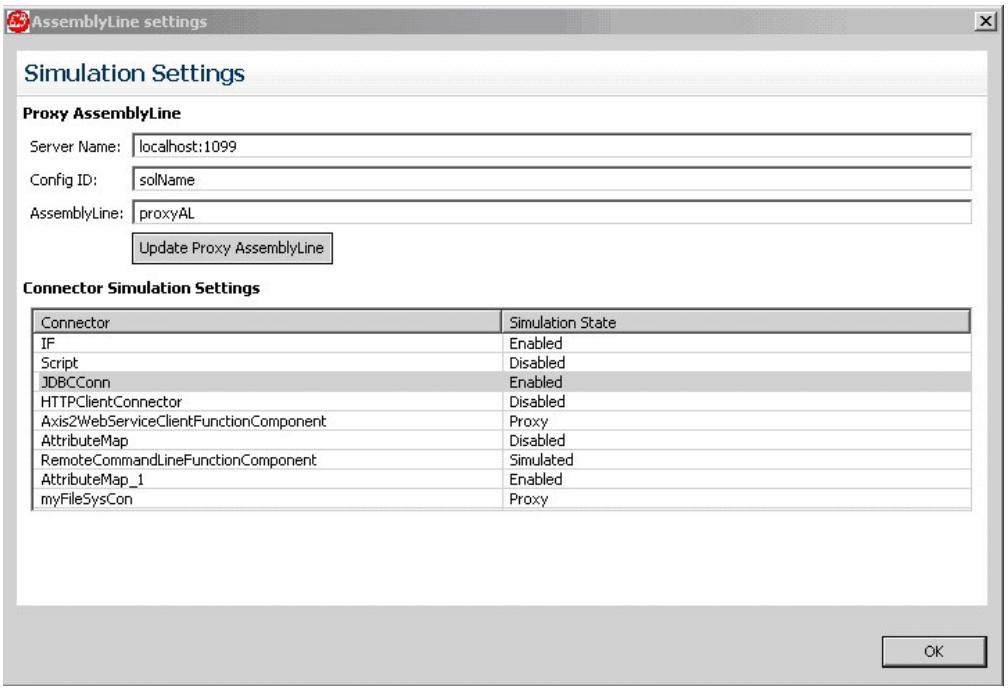


图 108. 模拟设置窗口

每个组件的模拟状态的配置通过使用“模拟设置”对话框来完成。该对话框配置由模拟状态设置为 *ProxyAL* 的组件使用的 ProxyAL。当单击**更新代理 AssemblyLine** 时，配置编辑器将在当前项目中创建新的 ProxyAL 或者将更新现有的代理 AssemblyLine。创建或更新的 ProxyAL 会作为模板提供，其结构基于您在“模拟设置”对话框中所做的配置。请注意，在这个过程中仅考虑代理 AssemblyLine 的名称。在模拟 AssemblyLine 的执行期间要考虑服务器名称和配置标识。如果已经存在带有对话框中指定的名称的 AssemblyLine，那么将仅添加新的分支，并且不会修改和除去任何旧的分支。这是因为某些旧的分支可能包含用户特定的配置。AssemblyLine 中的各个组件可以设置为以下某个状态:

已启用 这等价于在 AL 中以正常方式运行该组件（即组件按其正常运行的情况来执行）。

已禁用 这等价于禁用组件（即除了初始化之外，不为组件执行任何操作、挂钩和任何其他内容）。

已模拟

一般来说，如果 AL 未进行模拟，那么下面描述的所有组件的统计信息包含应已完成的操作的信息。由于在模拟期间未执行任何关键操作，所以无法预测执行关键操作的结果（成功或错误），因此统计信息将说明操作已成功完成。

- 处于 AddOnly 方式的连接器：按正常执行，仅跳过对 `connector.putEntry()` 方法可能不安全的调用和对 `override_add` 挂钩的调用。
- 处于 Update 方式的连接器：按正常执行，仅跳过对 `connector.modEntry()` 方法可能不安全的调用和对 `override_modify` 挂钩的调用。
- 处于 Delete 方式的连接器：按正常执行，仅跳过对 `connector.deleteEntry()` 方法可能不安全的调用和对 `override_delete` 挂钩的调用。
- 处于 Delta 方式的连接器：按正常执行，但由于该连接器依赖于对上述方法的调用，如果跳过上面的方法和挂钩，那么将模拟该连接器。

- 处于 Iterator 方式的带有“变化量”标记的连接器：按正常执行；对于这些组件，更改与上面描述的连接器相似，即跳过对 BTree 类（putEntry、modEntry 和 deleteEntry）的方法的可能不安全的调用。对于 CDDeltaTaskComponent，覆盖落实状态可禁用落实（“不自动落实”），但落实对于明确调用 CSDeltaTaskCoponent#commitDeltaState() 方法的用户仍然是可能的。
- 函数组件（FC）：按正常执行，但禁用“before_functioncall”、“after_functioncall”和“no_reply”挂钩的调用以及对 function.perform() 方法的调用。对这些挂钩的调用已禁用，因为它们与从禁用的执行方法返回的对象关联。唯一将完成的操作是更改将指示 FC 已成功地执行该操作的统计信息。

  还将执行输入和输出映射，但输入映射前面的实际条目是空白条目（这可能导致抛出异常，最好从依赖于“函数组件”返回的属性的输入映射中禁用每个简单属性映射，或添加检查来验证高级属性映射中检索到的属性的有效性；或者，您可以使用 NullBehaviour 机制来覆盖可能的错误）。
- 处于其他任何方式的连接器如常运行。

代理 处于该模拟状态的连接器将启动另一个 AL（已在“模拟”选项卡中指定），该 AL 将覆盖操作的可能不安全执行。此外部 AL 将在处于此模拟方式下的所有组件之间共享。并将通过名称与正在模拟的组件的名称匹配的其他操作执行该 AL。请参阅第 174 页的『代理 AssemblyLine 工作流程』。

已脚本化

用户定义的脚本将覆盖可能不安全的操作。处于该模拟方式的每个组件都有自己的挂钩，这与在组件之间共享 AL 的“代理”状态不同。请参阅第 175 页的『模拟脚本工作流程』。

通过使用以下方法，您能够检查 AL 是否正在模拟，还可以启用/禁用模拟：

- boolean AssemblyLine#isSimulating() 在挂钩中按如下方式使用：task.isSimulating();
- void AssemblyLine#setSimulating(boolean) 在挂钩中按如下方式使用：task.setSimulating(true);

通过使用以下方法，您能够检查每个组件的状态并对其进行动态设置：

- String AssemblyLineComponent.getSimulatingState() 在挂钩中按如下方式使用：ConnectorName.getSimulatingState();
- void AssemblyLineComponent.getSimulatingState(String) 在挂钩中按如下方式使用：ConnectorName.setSimulatingState(″Proxy″);

注：仅连接器和 FC 拥有上述完整模拟状态集合，其余的组件（和处于 Server 方式的连接器）仅有“已启用”和“已禁用”状态。

IBM Security Directory Integrator 认为某些 FC 是安全的，这些 FC 的缺省模拟状态为“已启用”，即在缺省情况下它们不会模拟并且将如常执行；这些是所有无法更改目标系统的 FC（因为它们根本没有进行连接）。安全的 FC 列表如下：

- CBE FC
- JavaToXML FC
- XMLToJava FC
- SDOToXML FC

- XMLToSDO FC
- 解析器 FC
- MemQueue FC
- JavaToSOAP FC
- SOAPToJava FC
- WrapSOAP FC

警告: 您还可您还可以通过明确编码对底层数据系统进行更改；但这超出了“模拟”方式的讨论范围。

此处未列出的其余任何 FC 都被认为是可能不安全的，并且它们的缺省模拟状态将设置为“已模拟”。

代理 AssemblyLine 工作流程

在模拟方式的上下文中，对代理 AL 的调用的工作方式与您使用 AssemblyLine 连接器驱动所选 AL 的方式相似，但是您也会看到一些重要的区别。

当组件处于代理模拟状态时，将禁用对特定操作的覆盖挂钩的调用。

下面的表显示了当连接器处于这些方式中的某个方式时或组件是函数组件时调用的方法。

表 10. 根据方式的方法调用

方式	方法
连接器: AddOnly	putEntry
连接器: Update	findEntry、modEntry 和 putEntry
连接器: Delete	findEntry 和 deleteEntry
连接器: Delta	findEntry、modEntry、putEntry 和 deleteEntry
连接器: Iterator	selectEntries 和 getNextEntry
连接器: CallReply	queryReply
连接器: Lookup	findEntry
函数组件	perform

当到达特定方法的调用时间时，且组件处于代理模拟状态，那么会将对该方法的调用委派到代理 AL。代理 AL 启动后，将把带有以下属性的操作条目传给该代理 AL:

- $operation – 该属性包含将执行的操作的名称。如果调用代理 AL 而不是组件的特定方法，那么该属性的值将与组件的名称相同。
- $method – 该属性包含将在正常情况下调用的方法的名称，但是由于组件处于代理模拟状态，并且处于某些方式（例如 Update）的组件在作实际修改（即 findEntry）之前会执行几个方法，因此代理 AL 必须识别将实施的方法。此 $method 属性只是告诉代理 AL 其所执行的方法，以便代理 AL 可以正确地处理操作。
- search – 如果 $method 属性为 findEntry，那么该属性可用。它的值是 SearchCriteria 类型的对象，并表示由用户定义的搜索条件。例如，如果组件处于代理模拟状态并且其方式为 Update，那么必须定义搜索条件以更新目标系统上正确的条目。由于模拟状态是代理，因此修改操作之前发生的实际查找操作将其执行委派给代理 AL。然后代理 AL 使用该搜索条件来执行正确的查找模拟。

- current – 该属性仅在 $method 属性为 modEntry 时可用。它的值是代表实际修改发生前在目标系统中找到的条目的条目对象。这是在 modEntry 方法之前执行的 findEntry 方法返回的条目。

当调用初始工作条目（IWE）时，将其传递到代理 AL。如果 $method 为 findEntry、selectEntry 或 getNextEntry，那么 IWE 是来自调用 AL 的工作条目的副本。在任何其他情况下，IWE 是从输出映射过程（又称为 *conn* 条目）检索到的条目。特别地，对于 deleteEntry 操作，IWE 是从先前的 findEntry 操作检索到的条目。

在完成代理 AL 执行之后并且 $method 是 findEntry，将检查结果条目以查找属性 *conn*。如果它是可用的，那么假定该属性包含从 findEntry 操作找到的所有条目，并将根据其值调用相应的挂钩（即 no_match 和 multiple_match）。如果未找到名为 conn 的属性，那么将代理 AL 执行的结果条目视为由 findEntry $method 找到的条目。从覆盖 selectEntries $method 的代理 AL 检索到的条目与调用 AL 的工作条目自动合并。将从模拟这些方法（这些方法期望诸如 findEntry、getNextEntry、queryReply 和 perform 的条目）的代理 AL 检索到的条目发送到已定义的输入映射。对于所有其他不期待返回结果的方法，将忽略来自代理 AL 的条目。

模拟脚本工作流程

模拟状态设置为“已编写脚本”的每个组件都在“模拟”选项卡中定义了模拟脚本（SS）。

此处向您显示的是可从 SS 直接使用的对象列表:

- *work* – 工作条目
- *conn* – 从查找操作检索或直接从 OutputMap 检索到的条目，或者为 null。
- *resEntry* – 如果正在模拟的操作需要返回结果，那么将此条目用作结果；如果不使用结果，那么它将为 null。
- *current* – 在将修改的目标系统中找到的条目，或为 null。
- *search* – 由用户定义的 SearchCriteria 对象，或为 null。
- *method* – 字符串对象，包含由 SS 覆盖的方法的名称。

下面的表说明了正在模拟的方法，它显示了将调用 SS 的时间。

表 11. 根据方式的方法调用

方式	方法
连接器: AddOnly	putEntry
连接器: Update	modEntry 或 putEntry（如果找不到该条目）
连接器: Delete	deleteEntry
连接器: Delta	modEntry、putEntry 或 deleteEntry（根据内部逻辑）
连接器: Iterator	getNextEntry
连接器: CallReply	queryReply
连接器: Lookup	findEntry
函数组件	perform

如果连接器处于 Server 方式且模拟状态为“已编制脚本”，那么它仍需要接收来自客户机的请求。如果要发送响应，那么将调用 SS。

为处于 Update、Delete 和 Delta 方式的连接器执行的内部查找操作将在内部执行，并且不允许从 SS 进行覆盖（例如对代理 AL 执行的覆盖）。

处于该模拟状态的连接器将不会执行操作的覆盖挂钩（如果有的话）。

第 5 章 Easy ETL

配置编辑器中的 EasyETL 透视图是一种查看 IBM Security Directory Integrator 配置的高度专用方法，旨在快速处理围绕简单任务的项目，以将数据抽取、变换和装入 (ETL) 到某种格式的数据库中。

EasyETL 透视图会显示 EasyETL 项目，并且使您能够运行、打开和创建新 ETL 项目。选择**窗口 > 打开透视图**，然后选择 Easy toETL 透视图可显示 ETL 透视图。

要通过此透视图启动 CE，请向 CE 命令行 (`ibmditk`) 添加 `-perspective com.ibm.tdi.rcp.perspective.etl`。

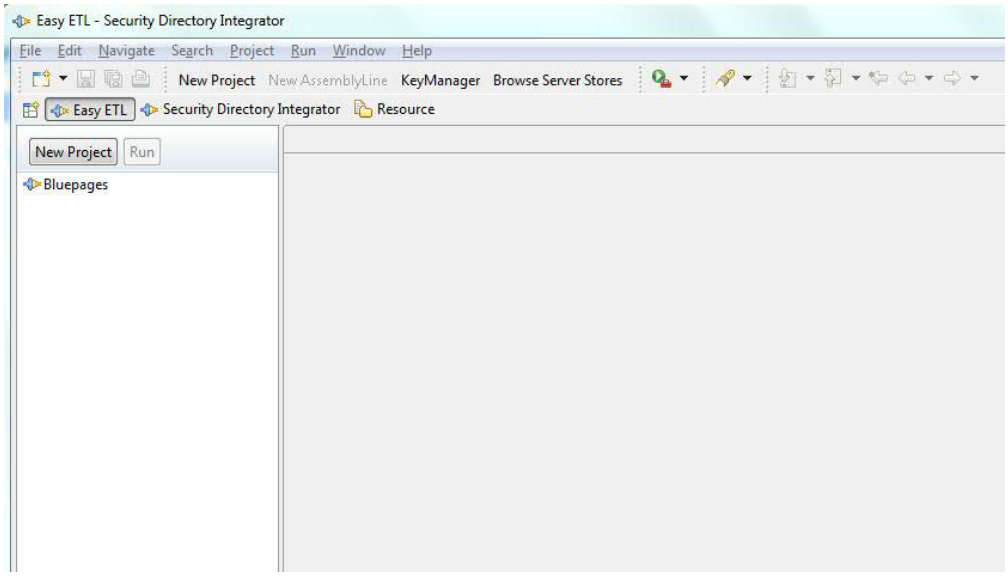


图 109. EasyETL 主窗口

使用**新建项目**按钮可以创建新 EasyETL 项目。EasyETL 项目是普通的 IBM Security Directory Integrator 项目，它有一个带两个连接器的 AssemblyLine。双击或选择项目并按 Enter 键将会打开 ETL 编辑器。

ETL 项目的上下文菜单包含下列项:

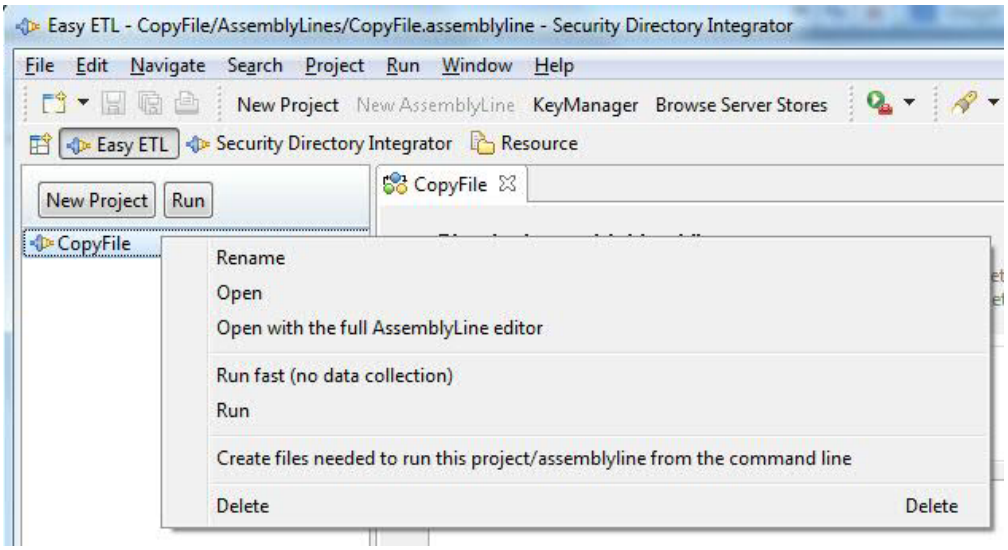


图 110. EasyETL 项目上下文菜单

- **打开** – 在编辑器中打开项目
- **使用完整 AssemblyLine 编辑器打开** – 在高级 AssemblyLine 编辑器中打开项目
- **快速运行** – 运行项目而不从 AssemblyLine 收集数据
- **运行**– 运行项目并在数据收集器视图中显示收集到的数据
- **创建文件 ...** – 生成从命令行运行项目所需的文件
- **重命名** – 重命名项目
- **删除** – 删除项目

EasyETL 编辑器通过表来显示两个连接器，在该表中会显示这两个连接器之间的映射。初始屏幕显示针对这两个连接器的选择均为空，如下图所示:

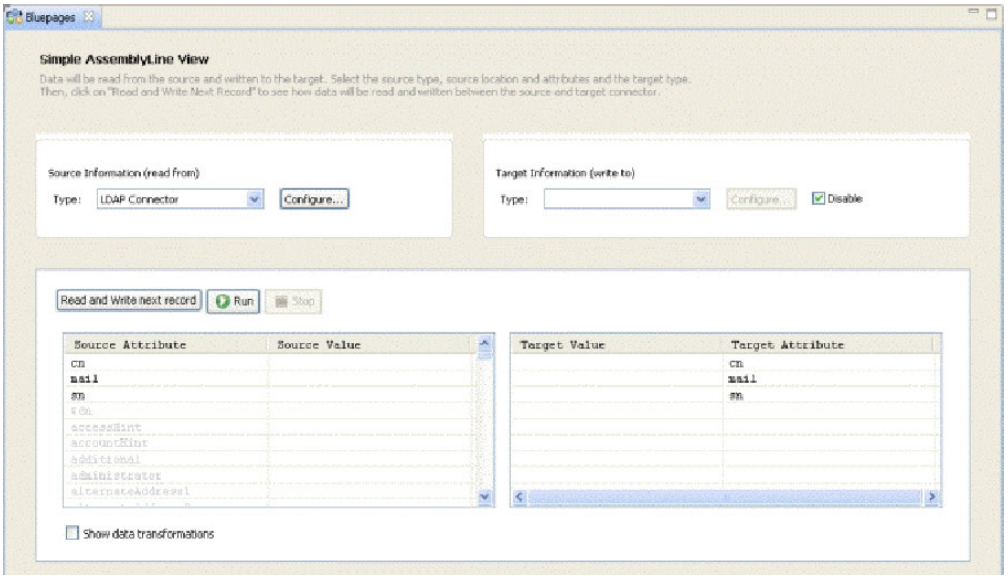


图 111. 初始 EasyETL 项目窗口

然后，通常通过选择源连接器进行启动。在类型下拉列表中有四个选项:

- 文件系统连接器
- LDAP 连接器
- 数据库连接器 (JDBC)
- 选择连接器...

前三个选项由于常用，因此为快速选择。最后一个选项，**选择连接器...** 会显示标准连接器选择对话框，可以在其中选择所需的任何连接器。不过，连接器列表仅限于对源 (Iterator) 和目标 (AddOnly) 连接器实施此方式的连接器。

一旦选定连接器，即可对其进行配置。配置对话框包含用于配置连接器和解析器（如果需要）的表单。此外，对于源连接器还会提供变化量屏幕。

如果选择“LDAP 连接器”，那么将显示以下窗口:

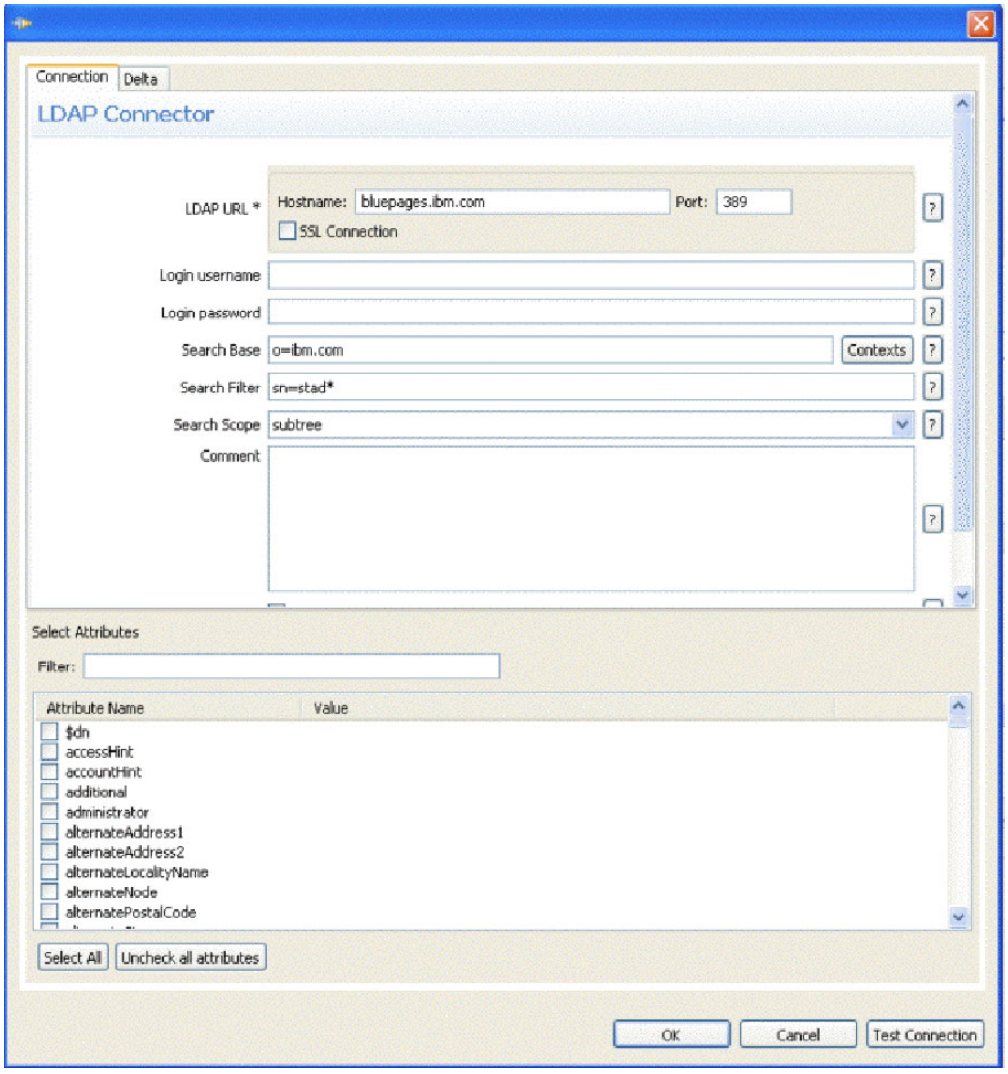


图 112. Easy ETL 中的 LDAP 连接器

在连接器中发现属性后，可以对要从连接器中读入的属性进行检查。对于目标连接器，由于映射基于输入连接器的属性，因此仅有模式项列表。

一旦模式和属性可用，便会在源属性列中将其显示。

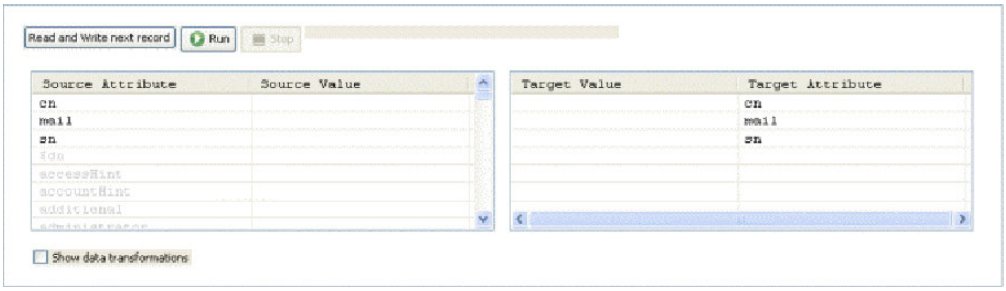


图 113. 输入/输出映射

灰显的项为未映射的属性。右键单击并选择**映射属性**可以对属性进行映射。您也可以双击或按 Enter 键以映射当前选择。在目标属性列中，可以单击并选择其他输出属性名称。反过来，可以对映射属性执行相同的操作，以将其返回到未映射属性列表。

要定制两个属性之间的映射，请选中**显示变换**复选框。这将在源表和目标表之间添加新表。

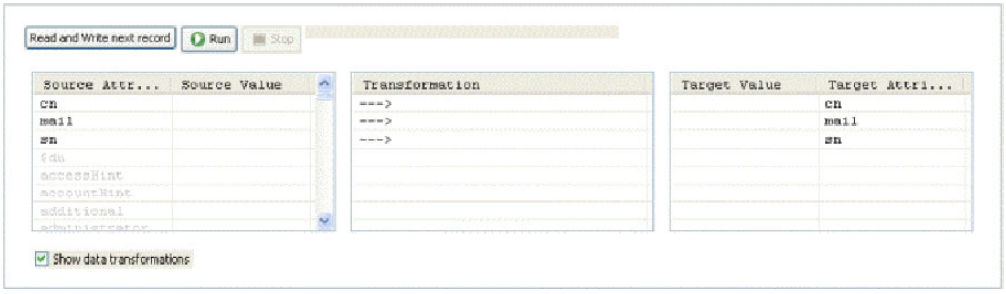


图 114. 带变换的输入/输出映射

所显示的变换表具有表示两个属性之间的逐字副本的箭头。双击变换项将会针对该映射显示 JavaScript 编辑器。

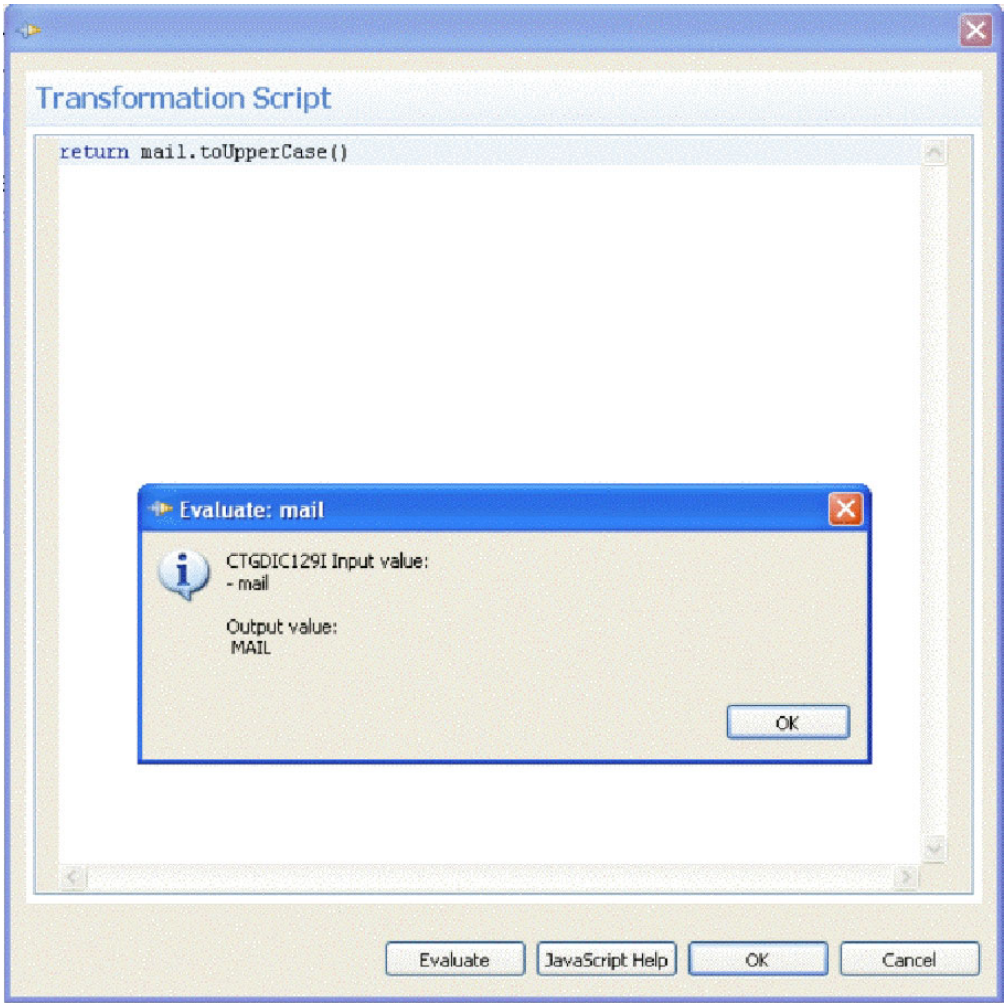


图 115. 变换脚本

输入将执行值的定制变换的脚本。请注意，源连接器中的所有映射属性都可用作顶级 bean。这意味着您可以直接引用 `cn` 而不是使用 `work.cn` 符号表示法。编辑器还会基于*读写下一条记录*操作所读取的内容感知 Java 类。在通过**求值**按钮测试脚本时，还会使用所读取的上一个条目，从而可以根据上图中显示的真实现场数据对脚本的求值进行测试。消息会显示变换脚本的输入值和结果（输出）。**JavaScript 帮助**按钮是访问 JavaScript 帮助页面的快速方法。

目标连接器具有标示为**禁用**的复选框。选中此复选框时，将会禁用输出连接器，并且改为将条目转储（在定制变换后）到控制台日志。

配置连接器后，此时可以运行 AssemblyLine 直至完成，或者通过 AssemblyLine 每次处理一条记录。逐步运行 AssemblyLine 时，该表将会反映读写到目标连接器的上一个条目。单击**运行**按钮时，AssemblyLine 将连续执行，直至其完成或按**停止**按钮为止。在执行期间按下停止按钮时，AssemblyLine 将中断并将控制归还给您。如果您在具有控制时按下停止按钮，那么 AssemblyLine 将终止。AssemblyLine 终止后，将会显示完成对话框，其中包含有关运行的一些统计信息:

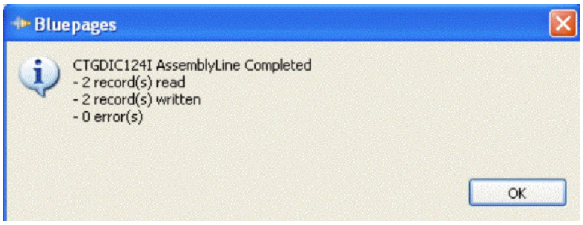

Bluepages
CTGDIC124I AssemblyLine Completed
- 2 record(s) read
- 2 record(s) written
- 0 error(s)
OK

图 116. 完成对话框

第 6 章 系统存储

“系统存储”会为持久存储处理 IBM Security Directory Integrator 的各种需求，缺省情况下将使用 Apache Derby RDBMS （以前称为 IBM Cloudscape）作为其底层的存储技术。

其他关系数据库（例如 IBM DB2）可用于保存系统存储。如果 Derby 数据库以联网方式运行，或者使用了多用户关系数据库系统，那么“系统存储”可供 IBM Security Directory Integrator 服务器的多个实例共享。如果 Derby 在 IBM Security Directory Integrator 服务器中以嵌入方式运行，那么它不能与其他服务器同时共享。

“系统存储”对 IBM Security Directory Integrator 组件实施三种类型的持久存储:

- 『用户特性存储』
- 第 184 页的『变化量存储』
- 沙箱表

每种存储都提供了自己的功能、内建行为以及可调用接口的集合，用户可以从它们的脚本访问该接口（例如，以持久保存用户的数据和状态信息）。

您可以为项目配置第 158 页的『系统存储设置』，方法是在 IBM Security Directory Integrator 导航器中选择此项目，然后选择**解决方案记录和设置**。然后选择**系统存储**选项卡。

您可以安装 JDBC 连接器来直接访问“系统存储”中的任何表，不过必须避免更改这些表中的数据，因为这样可能会导致解决方案出现故障。

警告: 如果您要在 IBM Security Directory Integrator 中以嵌入方式运行 Derby 而不是以联网方式作为服务器运行，那么务必先再次**关闭**数据库，然后再尝试测试或运行配置。因为配置编辑器启动服务器的单独实例（该服务器在自己的 JVM 中运行），所以“系统存储”不可用于此服务器。关闭 System Store Details 窗口也会断开到数据库的连接。

注: 不过“沙箱”功能也使用了“系统存储”技术，您为每个 AssemblyLine 指定了新的数据库目录。

用户特性存储

“用户特性存储”是一种用于维护与某个键值相关联的序列化 Java 对象的“系统存储”表。这是维护持久组件参数和属性（如**迭代器状态存储**）的地方，也是存储数据的地方。

例如，当您设置 Active Directory Change Detection 连接器的**Iterator 状态存储**参数时，将指定连接器用于保存和恢复 Iterator 状态的键值。如果您想迭代以首次（或上一次）更改的条目开始，只需删除“用户特性存储”中的“Iterator 状态存储”条目；也就是说，单击该参数旁边的**删除**。

您可以使用下面的系统调用来保持自己的对象:

system.setPersistentObject(*keyValue*,obj)
使用指定的 *keyValue* 将对象 **obj** 保存在“用户特性存储”中。如果对象保存成功则将其返回，否则函数返回 **null**。

system.getPersistentObject(*keyValue*)
从“用户特性存储”返回带有指定的 *keyValue* 的对象。如果找不到 *keyValue*，那么函数返回 NULL。

system.deletePersistentObject(*keyValue*)
删除“用户特性存储”中带有指定的 *keyValue* 的对象。此函数返回已删除的对象，如果找不到 *keyValue*，那么返回 NULL。

这些方法访问缺省的“用户特性存储”。

不过，你可以使用*存储工厂*创建和使用自己的存储。

如果您从“系统存储”窗口查看了“用户特性存储”，则会注意到它具有以下表定义:

键 唯一键（512 char）

条目 与键关联的对象

注: 任何要保持在“用户特性存储”中的对象必须是可序列化的。

变化量存储

变化量存储可在“系统存储”浏览器中的 **Delta Tables** 文件夹下面找到。每张表表示一个 **Delta Store** 参数设置（在迭代器的**变化量**选项卡中）。有许多类和方法可直接处理变化量存储，不过建议不要这么做。有关变化量功能的更多信息，请参阅标题为第 189 页的第 7 章，『变化量』的一节。

存储工厂方法

以下是可与“存储工厂”一起使用的方法的示例:

public static PropertyStore getDefaultPropertyStore () throws Exception;
返回缺省的特性存储。

public static PropertyStore getPropertyStore (String table) throws Exception; 返回以名称标识的“特性存储”。每次只出现一个给定名称的实例。

@param name
特性存储名称。

@return
与名称关联的特性存储对象。

public static String getSystemDatabaseURL ();
返回系统存储 JDBC URL。

public static Connection getConnection () throws Exception;
将连接对象返回至缺省数据库。

public static Connection getConnection (String database) throws Exception; 将连接对象返回到 AutoCommit 设置为 TRUE 的指定数据库。

@param database
数据库名称。

public static Connection getConnection (String database, boolean autoCommit) throws Exception;
将连接对象返回至指定数据库。

@param database
数据库名称。

@param autocommit
自动落实标志。

@return
指定数据库的连接对象。

public static boolean dropTable (Connection connection, String table);
删除与连接关联的数据库中的一张表。

@param connection
由 getConnection() 获取的连接对象。

@param table
要删除的表。

public static void verifyTable (Connection connection, String table, Vector sql) throws Exception;
验证数据库中的某张表是可存取的。

@param connection
由 getConnection() 获取的连接对象。如果为 null，那么获取到缺省表的连接。

@param table
要验证的表名。

@param sql
如果该表不存在，用于创建该表的 SQL 语句的向量。

public static Exception dropTable (String tableName);
删除缺省数据库中的一张表。

@param tableName
要删除的表的名称。

public static byte[] serializeObject (Object obj) throws Exception;
将对象序列化为字节数组。

@param obj
要序列化的对象。

@return
包含已序列化的对象的字节数组。

public static Object deserializeObject (byte[] array) throws Exception;
将字节数组逆序列化为 Java 对象。

@param array
带有已序列化的 Java 对象的字节数组。

@return
反序列化的 Java 对象。

属性存储方法

以下是可与“属性存储”一起使用的方法的示例:

public Object setProperty (String key, Object obj) throws Exception;
添加或更新“属性存储”中的值。如果执行了更新，则返回原有的值。

@param key
唯一标识。

@param obj
值。

@return
旧值（如果有更新）。

public Object getProperty (String key) throws Exception;
返回“属性存储”中的值。

@param key
唯一标识。

@return
存储中的值或 NULL（如果未找到）。

public Object removeProperty (String key) throws Exception;
除去“属性存储”中的值。

@param key
要除去的唯一标识。

@return
原有值或 **null**（如果键不在表中）。

UserFunctions（系统对象）方法

UserFunctions 类（例如，系统对象）定义了其他方法来获取或设置“系统属性存储”中的对象:

public Object getPersistentObject (String key) throws Exception;
此方法从缺省系统属性存储检索指定对象。

@param key
唯一键。

public Object setPersistentObject (String key, Object value) throws Exception;
此方法将指定对象存储到缺省系统属性存储中。

@param key
唯一键。

@param value
要存储的对象（必须是 Java 可序列化的）。

@return

旧对象（如果有的话）。

```
public Object removePersistentObject ( String key ) throws Exception;
```

此方法将指定对象从缺省系统属性存储中除去。

@param key

唯一键。

@return

旧对象（如果有的话）。

第 7 章 变化量

变化量引擎功能对于 Iterator 方式下的连接器可用。如果从 Iterator 的“变化量”选项卡中进行了启用，那么变化量引擎功能将使用系统存储器来拍摄进行迭代的数据的快照。成功读取的各条目会与名为“变化量存储器”的快照数据库相比较，以查看已更改的内容。基于所读取的条目和“变化量存储器”中存储的条目之间的差异，变化量引擎将创建名为“变化量条目”的新条目。此条目标有特殊的变化量操作码，用于指示已更改的内容及更改方式。

Delta 方式是一种连接器方式，通过此方式，连接器可以“理解”并使用变化量操作码。此方式下的连接器使用接收到的变化量条目的变化量操作码来确定需要应用到已连接系统的更改类型。Delta 方式支持所有类型的修改 – add、modify 和 delete。此方式用于促进不同系统之间的同步（例如，同步不同机器上的两个 LDAP 服务器）。

警告: 变化量引擎介绍了一种底层的本地存储库，该存储库用于存储数据的快照从而在同步过程中计算更改。正在扫描更改的数据源成为主从关系中的主数据源，然后至关重要的是对从数据源（例如，Delta 存储）进行的所有更改都是通过 Delta 机制而不是直接操作底层数据库表进行的。否则，IBM Security Directory Integrator 维护的 Delta 快照信息将不一致，而且 Delta 引擎将失败。

变化量功能

IBM Security Directory Integrator 中的变化量功能包括:

变化量条目

这是以特殊变化量操作码进行了标记的常规条目对象。这些代码描述更改的类型（*add*、*modify*、*delete* 或 *unchanged*），并且可以在不同级别（即条目、属性或属性值级别）上进行指定。

产生变化量条目的组件

- 变化量引擎 – 检测数据源中的更改。当数据源本身不提供对更改的便利访问途径（例如，changelog 或更改通知机制）时，这会非常有用。通过将数据源的当前状态与历史快照进行比较，可以检测更改。快照保存在名为“变化量存储器”的存储库中。变化量存储器在物理上由位于系统存储器中的多个变化量表组成。

  下次读取数据时，会将此数据与变化量存储器中已保存的数据相比较。计算更改信息后，连接器会创建并返回变化量条目。然后，此变化量条目可用于通过 *Update*、*AddOnly*、*Delete* 或 *Delta* 方式下的连接器来将已检测到的更改传输到其他已连接的系统。
- Change Detection 连接器 – 这些是指 LDAP 连接器、RDBMS Change Detection 连接器和 Domino Change Detection 连接器。
- LDIF 和 DSMLv2 解析器 – 读取或写入时，LDIF 和 DSMLv2 解析器支持在条目、属性和属性值级别进行变化量标记。

使用变化量条目的组件

- Delta 方式 – 此连接器方式会促进系统之间的同步解决方案。它可用于将所有类型的更改应用到已连接系统。Delta 方式是唯一需要（和使用）变化量条目的方式。此方式通过使用条目的变化量操作码来检测更改类型。
- 带有“计算更改”的 Update 连接器 – 如果接收到的条目标有变化量，那么已启用“计算更改”参数的 Update 方式下的连接器不触发“计算更改”逻辑。

变化量条目

变化量条目是具有常规条目的所有特性和功能的条目。除此之外，变化量条目还包含变化量操作码。这些操作码指示应用于条目的更改类型，即 add、delete、modify 或 unchanged。可以将变化量操作码附加到条目、属性和值以反映其特定更改。

概述

指定变化量操作码的过程称为变化量标记，变化量代码称为操作码和变化量标记。简单而言，变化量条目实际是标记了变化量的常规条目。以下是常规条目和变化量条目的示例。

常规条目:

```
{
 "#type": "generic",
 "#count": 3,
 "UserName":
  "#type": "replace",
  "#count": 1,"tanders",
 "FullName":
  "#type": "replace",
  "#count": 1,"Teddy Anderson",
 "id":
  "#type": "replace",
  "#count": 1,"66"
}
```

变化量条目:

```
{
 "#type": "modify",
 "#count": 3,
 "@delta.old": "{
 "UserName": "manders",
 "FullName": "Mary Anderson",
 "id": "66"
 }",
 "UserName": [
  "#type": "modify",
  "#count": 2,
  "tanders",
  "manders"
 ],
 "FullName": [
  "#type": "replace",
  "#count": 1,
  "Mary Anderson"
 ],
 "id":
  "#type": "unchanged",
  "#count": 1,"66"
}
```

变化量代码的求值过程自上向下或按条目级 → 属性级 → 属性值级顺序进行。较高级别的操作优先。

如果条目操作为 *delete*，那么将忽略所有其他变化量标记。如果条目操作为 *replace*、*modify* 或 *add*，那么将通过属性变化量标记继续求值。

如果属性标记为 *delete*、*add* 或 *replace*，那么将忽略其值的变化量标记。仅当属性标记为 *modify* 时，才将考虑属性值的变化量标记。这些变化量标记指示属性值可以具有不同的操作码（例如，将添加某些属性值，而删除其他属性值）。

属性值变化量标记在变化量标记上下文中具有下列含义:

- *add* – 将值添加到指定属性的值列表;
- *delete* – 将值从指定属性的值列表中除去;
- *replace* – 将值替换; 此为缺省变化量标记。

变化量条目由下列组件产生:

1. Iterator 方式下已启用 Delta 的连接器;
2. Change Detection 连接器;
3. 读取时的 LDIF 和 DSMLv2 解析器。

变化量条目供下列组件使用:

1. Delta 方式下的连接器;
2. 写入时的 LDIF 和 DSMLv2 解析器;
3. Update 方式下的连接器。

使用脚本获取和设置变化量操作码

变化量标记在条目、属性和属性值级别受支持。以下示例说明如何使用脚本来获取/设置条目操作:

```
var entryOper = work.getOperation();  //get Entry operations as string (e.g. 'add')
var entryOp = work.getOp();      //get Entry operations as char (e.g. 'a')

work.setOperation("modify");      //set Entry operation
work.setOp ('m');
```

如果要设置/获取属性的变化量标志，那么可以通过以下代码实现:

```
var attr = work.getAttribute("sn");   // get Attribute object

var attrOper = attr.getOperation();   // get delta operation as string (e.g. 'replace')
var attrOp = attr.getOper();      // get delta operation as char (e.g. 'r')

attr.setOperation("replace");     // set Attribute delta operation
attr.setOper('r');
```

可以使用以下脚本设置/获取属性值级别的变化量标记:

```
var attr = work.getAttribute("sn");   // get Attribute object

var valOper = attr.getValueOperation(0);// get value delta operation for first value
var valOp = attr.getValueOper(0);

attr.setValueOperation(1, "add");    // set value delta operation for second value
attr.setValueOper(1, 'a');
```

产生变化量条目

可以使用 Iterator 方式下连接器的变化量功能或适当解析器，或者通过 IBM Security Directory Integrator 的 Change Detection 连接器来生成变化量条目。

具体而言，标有变化量的条目由下列组件返回:

- Active Directory Change Detection 连接器
- Domino Change Detection 连接器
- IBM Security Directory Server Changelog 连接器
- RDBMS Change Detection 连接器
- Sun Directory Change Detection 连接器
- z/OS LDAP Changelog 连接器

  注: IBM Security Directory Integrator V7.2 及后续版本不支持 z/OS 操作系统。
- DSMLv2 Parser
- LDIF Parser

针对 Iterator 方式的变化量功能

Iterator 方式下的连接器可以生成变化量条目。此功能使用变化量引擎和变化量存储器来检测更改。

变化量引擎

通过变化量引擎，可以读取整个数据源，并且检测自上次执行此操作以来的更改。这样，即可检测新条目、已更改的条目甚至已删除的条目。对于某些数据源（如 LDIF 文件和 LDAP 服务器），IBM Security Directory Integrator 甚至可以检测条目中的属性和值是否已更改。只可以在处于 Iterator 方式的连接器上配置 Delta 设置。

变化量引擎通过在属于“系统存储器”的持久存储器中保留各条目的本地副本，可以知道是否已添加、更改或删除了条目或属性。此本地存储库称为*变化量存储器*并且包含*变化量表*。每次运行 AssemblyLine 时，变化量引擎都会将进行迭代的数据与其在变化量表中的副本相比较。如果检测到更改，那么连接器会返回*变化量条目*。

注: 请勿手动修改变化量存储器表。否则，变化量快照信息将变得不一致，并且变化量引擎将失败。

注: 在 IBM Security Directory Integrator V6.1 之前的版本中，变化量引擎处理期间写入到变化量存储器中的快照会立即落实。因此，即使处理“AL 流”部分失败，变化量引擎仍会将已更改的条目视为已处理。此限制可通过“连接器变化量”选项卡上的“落实”参数来解决。设置此参数可控制变化量引擎将所拍摄入局数据的快照落实到系统存储器的时间。

唯一属性名称

为使变化量机制能够唯一标识各条目，必须指定唯一属性以用作变化量键。此属性的值在所使用的数据源中必须唯一。通过在“唯一属性名称”参数中输入或选择属性名称，可以在连接器的“变化量”选项卡中指定变化量键。此属性必须位于 Iterator 的“输入映射”中，并且可以是从已连接系统读取的属性或计算属性（使用“属性映射”中的脚本）。

您也可以通过将属性以加号 (+) 分隔来指定多个属性:

```
LastName+FirstName+BirthDate
```

“唯一属性名称”参数中指定的至少一个属性必须包含值。指定了若干个属性时，它们的字符串值连接为一个字符串，然后该字符串成为唯一的 Delta 标识。如果为条目构建了变化量键，那么会跳过不具有值（例如空白或 NULL）的属性。

变化量存储

变化量存储器在物理上位于系统存储器中。它由一个变化量系统表 (DS) 和一个或多个变化量表组成。各变化量表用于其他已启用变化量的 Iterator 连接器的变化量存储器。

虽然可以通过 JDBC 连接器和系统存储器连接器访问变化量存储器表，但是不建议在未深入了解变化量引擎如何构造和处理这些表的情况下便对其进行更改。

Delta Table 结构

每个变化量表 (DT) 均包含变化量引擎为特定连接器处理的各条目的有关信息。变化量系统表 (DS) 以列表方式保留了当前正在由变化量存储器使用的所有变化量表。

- 变化量系统表 – 变化量系统表 (DS) 包含有关系统存储器中各变化量表 (DT) 的信息。 DS 的用途在于维护每个 DT 的序列计数器。DS 的结构如下:

表 12. Delta Systable 结构

列	类型	描述
标识	可变长字符串	DT 标识（名称）
序列标识	整数	从上一次运行起的序列标识
版本	整数	DS 版本（1）

- 变化量表 – 请求变化量存储器的各连接器都需要指定唯一变化量标识才能与连接器关联。这个标识还用作 Delta Table 在“系统存储”中的名称。Delta Table 的结构如下:

表 13. Delta Table 结构

列	类型	描述
标识	可变长字符串	用于标识条目的唯一值
序列标识	整数	条目的序号
条目	Long Varbinary	序列化的条目对象

Delta 过程

给定以上变化量存储器结构，序号用于定义不再属于源数据集的条目。每次运行 AssemblyLine 时，会从变化量系统表中读取特别是由连接器使用的变化量表的序号。然后，会将此序号递增，此递增值将用于在整个 AssemblyLine 执行期间标记已更新的条目。

变化量引擎进程以两次遍历来运行。

1. 读取 → 查找 → 比较 → 更新 → 设置当前序列标识
   a. Iterator 从输入数据源读取条目。
   b. Delta 进程使用唯一属性的值在变化量表中查找对应条目。

c. 如果找到匹配条目，那么 Delta 进程会比较各属性（及其值）以确定是否对该条目进行了修改。基于比较的结果，变化量引擎将返回标有相关操作码（*modify* 或 *unchanged*）的变化量条目。
   - Modify 条目 – 所读取的条目和变化量表中的对应条目被视为不同；该条目会在变化量表中进行更新
   - Unchanged 条目 – 所读取的条目和变化量表中的对应条目被视为等同。

d. 如果在变化量表中找不到匹配条目，那么会将该条目视为新条目：
   - Add 条目 – 该条目会添加到变化量表中。

e. 在 c. 和 d. 这两种情况下，都会使用用于当前 AssemblyLine 执行的序号来更新变化量表中的序号值。

2. 通过（序列标识 < 当前序列标识）检查数据 → 标记为“已删除”

一旦 Iterator 到达数据结尾，变化量引擎便会对变化量表进行第二次遍历来查找在第一次遍历期间未访问的那些条目。这些条目易于识别，因为其序号未使用当前序号进行更新。因此，变化量表中序号低于当前序号的任何条目均被视为已删除，并且会作为已删除条目返回。

注: 仅当涵盖全部输入数据的迭代成功完成后，才会发生此遍历。如果出于某种原因在该迭代期间发生错误，那么 AssemblyLine 不会将任何条目标记为已删除并返回，也不会有条目从变化量表中除去。这将不会影响原始数据源，并且在下次成功执行了 AssemblyLine 时，将正确处理已删除条目。

行锁定

在 Iterator 连接器的“变化量”选项卡以及“变化量函数组件”配置中会提供此参数。通过此参数，可以设置与变化量存储器数据库建立的连接所使用的事务隔离级别。设置更高的隔离级别可以通过使用行锁定和表锁定来减少事务异常（称为“脏读”、“不可重复读”和“幻读”）。此参数具有下列值:

READ_UNCOMMITTED

与 `java.sql.Connection.TRANSACTION_READ_UNCOMMITTED` 对应；指示可以进行脏读、不可重复读和幻读。此级别允许在落实由一个事务更改的某一行中的任何更改之前，让另一个事务读取此行（“脏读”）。如果回滚了任何更改，那么第二个事务将会检索到无效行。

READ_COMMITTED

与 `java.sql.Connection.TRANSACTION_READ_COMMITTED` 对应；指示脏读将被禁止；可以进行不可重复读和幻读。此级别仅禁止事务读取其中包含未落实更改的行。

REPEATABLE_READ

与 `java.sql.Connection.TRANSACTION_REPEATABLE_READ` 对应；指示脏读和不可重复读将被禁止；可以进行幻读。此级别禁止事务读取其中包含未落实更改的行，并且还禁止一个事务读取某一行，另一个事务修改该行，然后第一个事务重新读取该行，从而第二次获取不同值的情况（“不可重复读”）。

SERIALIZABLE

与 `java.sql.Connection.TRANSACTION_SERIALIZABLE` 对应；指示脏读、不可重复读和幻读都将被禁止。此级别包含 TRANSACTION_REPEATABLE_READ 中的禁令，并且进一步禁止一个事务读取满足 WHERE 条件的所有行，另一个事

务插入满足 WHERE 条件的行，然后第一个事务重新读取同一条件，从而检索第二次读取中的其他“幻像”行的情况。这通常是最慢但最安全的选项，并且是**行锁定**参数的缺省值。

有关事务隔离级别的更多信息，请参阅 java.sql.Connection 接口的联机文档: http://docs.oracle.com/javase/1.6.0/docs/api/java/sql/Connection.html。

每个数据库服务器会设置一个缺省事务隔离级别；Apache Derby、Oracle 和 Microsoft SQL Server 的缺省值为 TRANSACTION_READ_COMMITTED。但是，在使用变化量组件（即 Iterator 连接器中的变化量功能或者变化量函数组件）时，**行锁定**参数的缺省值 SERIALIZABLE 将覆盖上述值。

部分数据库服务器可能不支持所有事务隔离级别，因此请参见特定数据库文档，以获取有关受支持事务隔离级别的准确信息。

注: 事务隔离级别由数据库服务器本身针对与数据库建立的每个连接进行保留。因此，当变化量组件（**事务隔离级别**设置为 REPEATABLE_READ 或 SERIALIZABLE，并且**落实**参数设置为 `On Connector Close`）启动其事务时，将阻塞所有其他尝试修改相同数据的查询。这意味着需要修改相同数据的其他组件将必须等待，直至第一个组件在终止时落实其事务为止。此等待可能会导致已发出的 SQL 查询超时并使数据保持未修改。

此外，如果组件的**落实**参数设置为 `No autocommit`，那么应通过使其他组件将不会永远等待执行修改的方式来手动落实事务。

仅检测或忽略特定属性中的更改

参数**属性列表**和**更改检测方式**会配置变化量引擎的功能，使其仅检测特定属性而不是接收到的所有属性中的更改。

属性列表参数是以逗号分隔的属性的列表，将会受到**更改检测方式**影响。此**更改检测方式**参数指定将如何处理这些属性中的更改。它具有三个值:

IGNORE_ATTRIBUTES
（“忽略下列属性的更改”）– 在计算更改过程中将忽略**属性列表**参数中指定的每个属性的更改。

DETECT_ATTRIBUTES
（“检测下列属性的更改”）– 此选项具有相反效果 – 将仅检测到**属性列表**参数中列出的属性的更改。

DETECT_ALL
（“对更改检测使用所有属性”）– 此选项指导变化量引擎检测所有属性的更改。在选择了此选项时，由于无需任何受影响属性列表，因此会禁用**属性列表**参数。

示范用例

使用变化量引擎时，有时接收到的条目会包含您视为不重要并希望忽略的属性。在此类情况下，这些属性不得影响变化量计算的结果，因为如果若干个条目仅通过这些属性来区分，那么会导致对变化量存储器表进行不必要的更新。

针对此情况的解决方案是使用**属性列表**和**更改检测方式**参数。

以下是示例方案，其中两个 AssemblyLine 从 LDAP 服务器的两个副本接收 changelog 条目，并且这些更改会应用到一个变化量存储器。为对此进行说明，将使用以下示例 changelog 条目:

Entry1:

```
Entry attributes:
 targetdn (replace): 'cn=Niki,o=IBM,c=us'
 changetime (replace): '20071015094646'
 $dn (replace): 'changenumber=78955,cn=changelog'
 ibm-changeInitiatorsName (replace): 'CN=ROOT'
 changenumber (replace): '78955'
 objectclass (replace): 'top' 'changelogentry' 'ibm-changelog'
 changetype (replace): 'modify'
 cn (replace): 'Niki' 'Niky'
 changes (replace): 'replace: cn
       cn: Niki
       cn: Niky
       -
       '
```

Entry2:

```
Entry attributes:
 targetdn (replace): 'cn=Niki,o=IBM,c=us'
 changetime (replace): '20071015094817'
 $dn (replace): 'changenumber=10076,cn=changelog'
 ibm-changeInitiatorsName (replace): 'CN=ROOT'
 changenumber (replace): '10076'
 objectclass (replace): 'top' 'changelogentry' 'ibm-changelog'
 changetype (replace): 'modify'
 cn (replace): 'Niki' 'Nikolai'
 changes (replace): 'replace: cn
       cn: Niki
       cn: Nikolai
       -
      '
```

Entry3:

```
Entry attributes:
 targetdn (replace): 'cn=Niki,o=IBM,c=us'
 changetime (replace): '20071037454817'
 $dn (replace): 'changenumber=112,cn=changelog'
 ibm-changeInitiatorsName (replace): 'CN=ADMIN'
 changenumber (replace): '112'
 objectclass (replace): 'top' 'changelogentry' 'ibm-changelog'
 changetype (replace): 'modify'
 cn (replace): 'Niki' 'Nikolai'
 changes (replace): 'replace: cn
       cn: Niki
       cn: Nikolai
       -
          '
```

已修改属性以**粗体**标记，可以忽略的属性以*斜体*标记。比较接收到的条目和已存储的条目时，将不考虑已忽略的属性（如 changenumber 和 changetime 等等）。因此，必须在**属性列表**参数中列出这些属性。为指明要忽略这些属性，需要将**更改检测方式**参数设置为 Ignore changes for the following Attributes。

以下是 AssemblyLine 接收条目时的工作流程:

1. 当 AL1 接收 Entry1 时，会将其返回为 *modify* 并保存在变化量存储器表中。

2. 当 AL2 接收 Entry2 时，其 changetime、$dn、bm-changeInitiatorsName 和 changenumber 属性会进行更改但将被忽略。不过，cn 和 changes 属性也会修改，因此产生的变化量条目将标记为 *modify* 并保存在变化量存储器表中。
3. 当 AL2 接收 Entry3 时，其 changetime、$dn、bm-changeInitiatorsName、changenumber 属性会进行更改但将被忽略。其余属性等同，因此产生的变化量条目将标记为 *unchanged* 并返回到 AssemblyLine（仅当选中了**返回未更改项**参数时）或被跳过。返回的变化量条目将与接收到的 Entry3 相同。在此情况下，将不更新变化量存储器。如果未使用“属性列表”和“更改检测方式”参数，那么最后的 Entry3 会标记为 *modify* 并保存在变化量存储器中。

Change Detection 连接器

Change Detection 连接器利用已连接系统中的信息来检测更改，并且会根据连接器的不同，在 Iterator 或 Server 方式下使用。例如，Iterator 方式用于许多 Change Detection 连接器，如 LDAP、RDBMS、Active Directory 和 Notes/Domino 的 Change Detection 连接器。这些连接器旨在以共同方式工作，以及为公共设置提供相同的参数标签。

IBM Security Directory Integrator 中的 Change Detection 连接器包括:

- IBM Security Directory Server Changelog
- AD Change Detection（Active Directory）
- Domino Change Detection
- Sun Directory Change Detection（openLDAP、SunOne、iPlanet 等）
- RDBMS Change Detection（DB2、Oracle、SQL server 等）
- z/OS LDAP Changelog

注: IBM Security Directory Integrator V7.2 及后续版本不支持 z/OS 操作系统。

Delta 引擎功能可报告多个级别的特定更改，直至属性值级别。通过 LDIF 或 DSMLv2 解析器解析时，还可进行 AttributeValue 级别的变化量标记。IBM Security Directory Server Changelog 连接器、Sun Directory Change Detection 连接器和 z/OS LDAP Changelog 连接器可在内部使用 LDIF 解析器，因此，这些连接器也支持 AttributeValue 级别的变化量标记。其他 Change Detection 连接器仅限于报告是否添加、修改或删除了整个条目。有关特定组件的变化量标记支持的更多信息，请参见参考中该组件的特定描述。

某些情况下，长期运行的 AssemblyLine 可能需要多次处理相同的条目。这些条目将具有重复的变化量键并将导致 AssemblyLine 抛出异常。如果希望允许变化量键的重复，可以选中 Iterator 的“变化量”选项卡中的**允许重复变化量键**复选框。这意味着重复条目可由具有已启用变化量的 Iterator 连接器或者 Change Detection 连接器和 Delta 方式连接器的 AssemblyLine 进行处理。

注: 例如，可以拥有包含多个 Changelog 连接器和 Delta 方式连接器的 AssemblyLine。在这种情况下，如果 Delta 方式连接器指向作为 Changelog 连接器的同一底层系统，那么 Delta 操作可以再次触发 Changelog。由于没有方法可以区分新接收的更改和由“变化量”引擎触发的更改，所以应该认真考虑场景，以避免进入死循环。

由变化量引擎计算的变化量信息存储在“工作条目”对象中，并且根据所使用的 Change Detection 组件或功能部件，可以存储为*条目级*、*属性级*或*属性值级*操作码。

使用变化量条目

Delta 方式下的连接器和“计算更改”逻辑中 Update 方式下的连接器可对 AssemblyLine 中的变化量条目执行操作（“使用”）。

Delta 方式在下列连接器中可用:

- JDBC 连接器
- DSMLv2 SOAP 连接器
- JNDI Connector
- LDAP 连接器

Delta 方式连接器

设计 Delta 方式可以基于变化量操作代码通过提供对已连接系统的递增修改来简化数据更改的应用。递增修改意味着仅写入已更改的特定值。

首先，Delta 方式处理所有类型的变化量: 添加、修改和删除。Delta 方式需要接收变化量条目才能运行。因此，在 Delta 方式下使用连接器时，必须将其与产生变化量条目的组件相结合；这些组件是 Iterator 连接器（已启用变化量）、Change Detection 连接器或者使用 LDIF 或 DSMLv2 解析器的连接器。例如，可以通过仅使用两个连接器将两个系统同步: 一个是位于“馈入”部分中的 Change Detection 连接器，用于选取更改；另一个是 Delta 方式下的连接器，用于将这些更改应用于目标系统。

而且，Delta 方式将把 Delta 信息应用到目标系统本身所支持的最低级别。实现此目标的方法是首先检查连接器接口，以查看受数据源支持的递增修改级别。如果使用的是 LDAP 目录，那么 Delta 方式将执行属性值添加和删除。在传统 RDBMS (JDBC) 上下文中，删除列值再将其添加没有意义，因此对于该属性，会将这作为值替换进行处理。

此外，对于支持此功能的数据源，递增修改会通过 Delta 方式来自动处理。如果数据源提供了优化的调用来处理递增修改，并且连接器接口支持这些调用，那么 Delta 方式将使用这些调用。另一方面，如果连接的系统不提供“智能”变化量更新机制，Delta 方式将尽可能模拟这些机制，执行预更新查找（比如 Update 方式），更改计算以及对检测到的更改的后续应用。

注: 支持递增修改的连接器只有 LDAP 连接器，因为 LDAP 目录提供此功能。

IBM Security Directory Integrator 中的变化量功能旨在不仅使用提供更改检测机制的数据源，还使用平面文件之类的方法来促进同步解决方案。下图显示了使用 Iterator 和 Delta 方式下的连接器的此类同步解决方案。 Iterator 连接器从数据源中读取条目。所读取的条目会与先前迭代时变化量存储器中保存的条目相比较。比较的结果是通过用于定义所作更改（即 add/delete/modify）的变化量操作码指定的变化量条目。然后，变化量条目由 Delta 方式下的连接器用来将检测到的更改应用于目标系统。

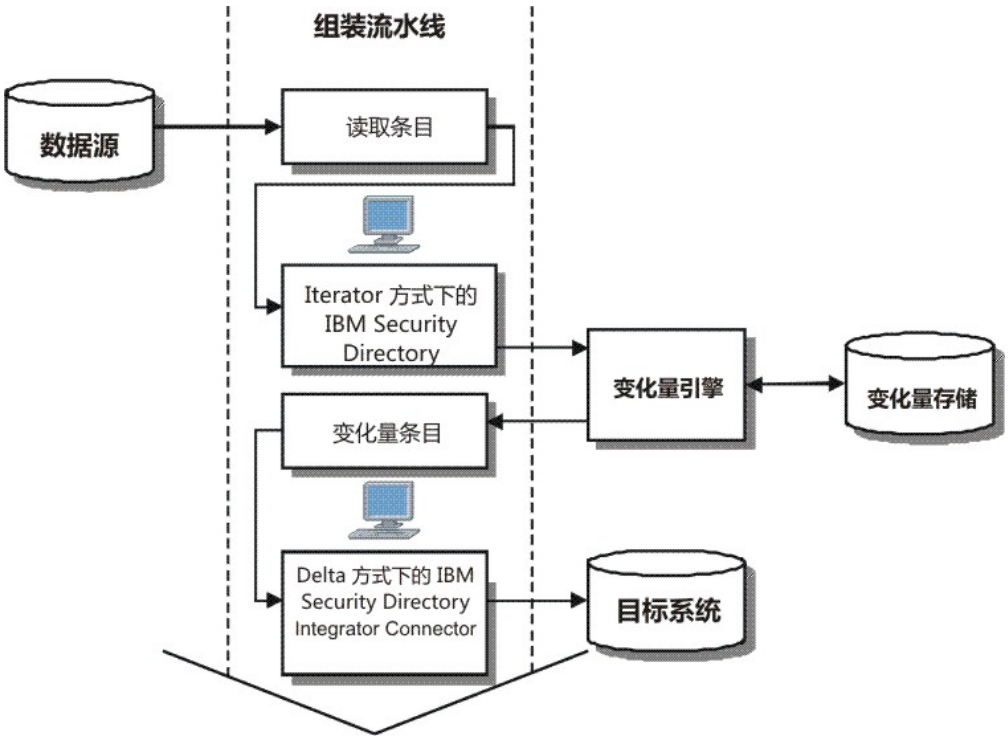


图 117. 使用变化量功能的同步 AssemblyLine

AssemblyLine 执行的结果是数据源中包含的数据与目标系统中的数据同步

Update 方式和变化量条目

Update 方式下的连接器另外具有一项名为“计算更改”的更改控制功能。此功能可用于变化量条目。

可以在 Update 方式连接器中启用“计算更改”功能，方法是在“连接器详细信息”窗格内选中具有该名称的复选框。当打开时，连接器会将其条目与已连接系统中的实际条目相比较。如果两个条目等同，那么连接器将不执行任何更新。因此，通过“计算更改”选项，Update 方式下的连接器可以跳过不必要的更新。这对于包含相对繁重更新操作的数据源非常有用。

不过，当具有已启用的**计算更改**参数的 Update 方式连接器接收变化量条目时，将不触发“计算更改”逻辑；它将使用指定给变化量条目的变化量操作码来将必要的更新应用到已连接系统。

示例

简单变化量示例

下方的指示信息将显示如何利用上文已讨论的一些变化量功能。

设置已启用变化量引擎功能的文件系统连接器。使这个连接器对一个可在文本编辑器中方便地进行编辑的简单 XML 文档执行迭代操作。例如:

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
    <Entry>
        <Telephone>
```

```
            <ValueTag>111-1111</ValueTag>
            <ValueTag>222-2222</ValueTag>
            <ValueTag>333-3333</ValueTag>
        </Telephone>
        <Birthdate>1958-12-24</Birthdate>
        <Title>Full-Time SDI Specialist</Title>
        <uid>jdoe</uid>
        <FullName>John Doe</FullName>
    </Entry>
</DocRoot>
```

请确保使用特殊的“映射全部”属性映射字符，即星号（*）。这是您在映射中唯一需要的属性，以确保返回的所有属性都映射至工作条目对象中。

现在通过以下代码添加脚本组件:

```
// Get the names of all Attributes in work as a String array
var attName = work.getAttributeNames();

// Print the Entry-level delta op code
task.logmsg(" Entry (" +
work.getString( "FullName" ) + ") : " +
work.getOperation() );

// Loop through all the Attributes in work
for (i = 0; i < attName.length; i++) {

 // Grab an Attribute and print the Attribute-level op code
 att = work.getAttribute( attName[ i ] );
 task.logmsg("    Att ( " + attName[i] + ") : " + att.getOperation() );

 // Now loop through all the Attribute's values and print their op codes
 for (j = 0; j < att.size(); j++) {
  task.logmsg( "      Val (" +
                 att.getValue( j ) + ") : " +
                 att.getValueOperation( j ) );
 }
}
```

第一次运行这个 AL 时，您的脚本组件代码将创建如下的日志输出:

```
12:46:31   Entry (John Doe) : add
12:46:31      Att ( Telephone) : replace
12:46:31        Val (111-1111) :
12:46:31        Val (222-2222) :
12:46:31        Val (333-3333) :
12:46:31      Att ( Birthdate) : replace
12:46:31        Val (1958-12-24) :
12:46:31      Att ( Title) : replace
12:46:31        Val (Full-Time SDI Specialist) :
12:46:31      Att ( uid) : replace
12:46:31        Val (jdoe) :
12:46:31      Att ( FullName) : replace
12:46:31        Val (John Doe) :
```

因为在先前的空“Delta 存储”中找不到该条目，所以它将在条目级标记为 new。而且，它的每个属性都具有 replace 代码，这表示所有值都进行了更改（这很有意义，因为 Delta 表明这是新数据）。

对您的 XML 文件进行以下更改:

1. 将最后一个电话号码值更改为 333-4444。
2. 删除 Birthdate。
3. 添加新的 Address 属性。

您的结果配置应该类似于以下内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
    <Entry>
        <Telephone>
            <ValueTag>111-1111</ValueTag>
            <ValueTag>222-2222</ValueTag>
            <ValueTag>333-4444</ValueTag>
        </Telephone>
        <Title>Full-Time SDI Specialist</Title>
        <uid>jdoe</uid>
        <FullName>John Doe</FullName>
        <Address>123 Willowby Lane</Address>
    </Entry>
</DocRoot>
```

再次运行 AL。这一次您的日志输出应该类似于这样:

```
13:53:22   Entry (John Doe) : modify
13:53:22      Att ( Telephone) : modify
13:53:22        Val (111-1111) : unchanged
13:53:22        Val (222-2222) : unchanged
13:53:22        Val (333-4444) : add
13:53:22        Val (333-3333) : delete
13:53:22      Att ( Birthdate) : delete
13:53:22        Val (1958-12-24) : delete
13:53:22      Att ( uid) : unchanged
13:53:22        Val (jdoe) : unchanged
13:53:22      Att ( Title) : unchanged
13:53:22        Val (Full-Time SDI Specialist) : unchanged
13:53:22      Att ( Address) : add
13:53:22        Val (123 Willowby Lane) : add
13:53:22      Att ( FullName) : unchanged
13:53:22        Val (John Doe) : unchanged
```

现在条目已标记为 *modify* 并且属性反映了对它们中的每个条目所做的修改。可以看到 Birthdate 属性已标记为 *delete*，而 Address 标记为 *add*。这正是您对“输入映射“使用特殊的“映射全部”字符的原因。如果您只映射了该 XML 文档第一版中存在的属性，则当 Address 出现在输入中时，我们将无法检索到它。

请特别注意 Telephone 属性下标记为 *modify* 的最后两个值条目。对此属性的其中一个值的更改先后产生了两个变化量项：值 *delete* 和 *add*。

要在先前版本的 IBM Security Directory Integrator 中构建数据同步 AssemblyLine，必须编写脚本以便处理流程控制。虽然您可能会从更改组件或功能部件中收到 *adds*、*modifies* 和 *deletes*，但是只能将连接器设置为两种必需输出方式之一：Update 或 Delete。

因此，您需要两个指向同一目标系统的连接器，并且将脚本放入各连接器的 Before Execute 挂钩中，以在条目的操作码与此组件的方式不匹配时忽略该条目；或者也可以具有处于 *Passive* 状态的单个连接器（Update 或 Delete 方式），然后从您检查操作码的脚本代码控制其执行。这仍意味着即使您在已修改条目的情况下知道所更改的内容，Update 方式连接器仍然会先读入原始数据，然后再将更改回写到数据源。当您更改的只是包含数千个值的多值组相关属性中的单个值时，这可能会造成不必要的网络或数据源流量。

更多示例

请转至 IBM Security Directory Integrator 安装版的 `root_directory/examples/` 目录。

- 在子目录 `deltas` 中，您将会找到演示变化量功能的简单 IBM Security Directory Integrator 配置。仅当此演示使用 MS Access 数据库和 JDBC:ODBC 网桥时，才能在 MS Windows 系统 (Windows 2000) 上运行。如果使用其他平台，那么必须创建自己的数据库，并且为此数据库配置连接器的 JDBC 设置。
- 在子目录 `delta_tagging` 中，您将找到演示如何将在值级别标记的条目转换为常规条目以及将常规条目转换为变化量条目的示例。

第 8 章 IBM Security Directory Integrator 仪表板

使用 IBM Security Directory Integrator 仪表板安装、配置、部署和监视数据集成解决方案。

还可以使用仪表板创建和配置 EasyETL 解决方案（ETL 代表抽取，变换和装入），用于简单的数据集成。 EasyETL 解决方案包含一个组装流水线，它有一个源连接器和一个目标连接器。

介绍 IBM Security Directory Integrator 仪表板

仪表板是一个 Web 应用程序，它是使用开放式服务网关协议 (OSGI) 框架开发的。它用于在服务器上通过 RESTful（表述性状态转移）接口配置和管理数据集成解决方案。在仪表板中，您可以:

- 上载现有数据集成解决方案并将其作为普通解决方案安装，或另存为模板
- 配置数据集成解决方案以反映特定于您的本地环境的更改
- 添加调度以在指定时间运行组装流水线
- 构建带有一个源连接器和一个目标连接器的简单组装流水线
- 浏览所选连接器的内容
- 部署、监视和审计数据集成解决方案的进程
- 查看服务器详细信息、服务器日志和系统存储数据
- 通过设置过滤条件来过滤日志文件数据，从而仅抽取出所需信息
- 以图形形式查看组装流水线执行历史记录
- 创建包含有关组装流水线执行状态的数据的报告，通过电子邮件自动发送给用户

IBM Security Directory Integrator 仪表板的优点

优点有:

- 促进数据集成解决方案的简化和快捷部署
- 在不使用配置编辑器的情况下创建和配置简单集成
- 使用仪表板 RunReport 功能部件生成组装流水线历史记录的调度电子邮件报告并发送。可以使用该报告满足部署的高可用性/故障转移需求。
- 促进与 Eclipse 开发环境的集成
- 确保高效管理 AssemblyLine 执行

访问仪表板应用程序

您可以从浏览器或配置编辑器访问仪表板 Web 应用程序。

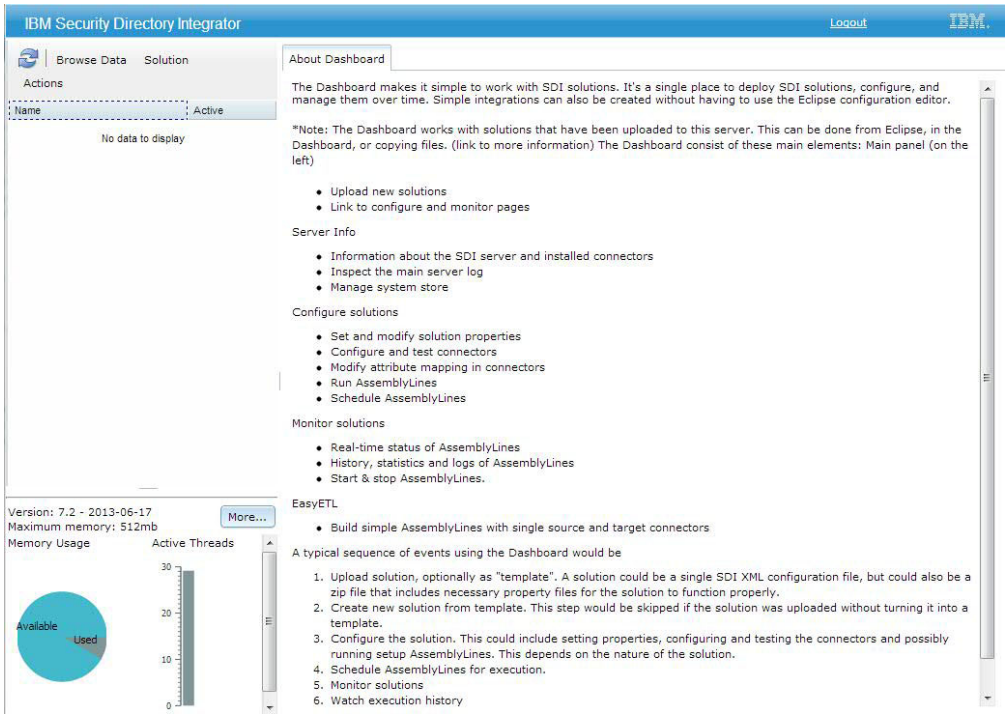
开始之前

访问仪表板之前，请先启动 IBM Security Directory Integrator Server。

从浏览器打开

过程

在浏览器中，转到 https<*host name*>:<*rest-api-port*>/dashboard。将显示仪表板主页。



注: 缺省情况下，您可以在本地主机上访问仪表板。要对用户进行认证，请将 IBM Security Directory Integrator 属性文件设置为添加可供远程和本地访问的 LDAP 用户名和密码。

从 Windows“开始”菜单打开

过程

从 Windows **开始**菜单中选择**开始 > 所有程序 > IBM Security Directory Integrator V7.2 > 打开仪表板**。 此时将显示“IBM Security Directory Integrator 仪表板”主页。

表 14. IBM Security Directory Integrator 仪表板菜单选项

菜单选项		描述
浏览数据		使用此选项可配置所选连接器并浏览其内容。
解决方案	**监视**	使用此选项可跟踪组装流水线执行进度并查看解决方案的执行历史记录。
	启动	使用此按钮可启动所选解决方案。
	停止	使用此按钮可终止运行的解决方案。
	配置	使用此按钮可打开所选解决方案供修改。
	删除	使用此选项可从服务器删除所选解决方案。
	解锁解决方案	使用此按钮可解锁解决方案。

表 14. IBM Security Directory Integrator 仪表板菜单选项 (续)

菜单选项		描述
操作	**浏览数据**	使用此选项可配置所选连接器并浏览其内容。
	显示系统日志	使用此选项可查看系统日志 (ibmdi.log) 的内容。
	创建解决方案	使用此选项可选择模板并创建数据集成解决方案。
	上载解决方案	使用此选项可将现有解决方案上载到 IBM Security Directory Integrator Server。
	显示服务器详细信息	使用此选项可查看有关 IBM Security Directory Integrator 的信息（例如，版本、已安装的组件和其他服务器信息）。

用于远程访问的 Internet Explorer 设置

添加所需配置设置，以便从远程系统中启用了 Internet Explorer Enhanced Security Configuration (IE ESC) 的 Internet Explorer 浏览器访问仪表板。

缺省情况下，IE ESC 将阻止所有在 Web 页面上运行的脚本。在 Web 页面上显示任何内容前，仪表板将装入多个脚本。因此，在您打开仪表板时，启用了 IESC 的 Internet Explorer 浏览器将显示一个空白页面。要访问页面，您必须向 Internet Explorer 的安全列表中添加托管该仪表板的站点。

1. 从 **Internet Explorer** 菜单中，单击**工具** > **Internet 选项**。
2. 单击**安全**选项卡。
3. 单击**受信任的站点**。
4. 单击**站点**。
5. 在**将该网站添加到区域**字段中，输入仪表板的 URL。例如：`https://mydashboard.com/dashboard/*`。
6. 单击**添加**。
7. 单击**关闭**，然后单击**确定**以关闭此页面并保存设置。
8. 重新启动 Internet Explorer 浏览器。
9. 在 Internet Explorer 浏览器中访问仪表板。

上载数据集成解决方案

使用仪表板的“上载解决方案”窗口可上载现有数据集成解决方案，从而将其安装在 IBM Security Directory Integrator Server 上。还可以将上载的解决方案作为模板保存。

关于此任务

仪表板可用于上载现有的 IBM Security Directory Integrator 数据集成解决方案（XML 配置文件），从而将其安装在服务器上。在部署前可以使用必需属性修改解决方案。还可以上载现有解决方案以将其另存为模板。保存的模板可用于创建同一集成的许多实例。

过程

1. 在“仪表板”窗口的导航窗格中，单击**操作 > 上载解决方案**。
2. 在“上载解决方案”窗口中定义下列设置。

选项	描述
浏览	浏览要上载的现有解决方案
作为模板上载	将上载的解决方案作为模板保存。 **注**: 已保存的模板未安装在 IBM Security Directory Integrator Server 上（配置目录）。
覆盖现有解决方案	覆盖现有的已安装解决方案或模板
解决方案名称	指定上载的解决方案的名称。 **注**: 还可在“上载解决方案”窗口中查看当前安装的解决方案和模板解决方案的名称。

3. 单击**确定**。

创建数据集成解决方案

使用仪表板的“创建解决方案”窗口可使用各种选项创建数据集成解决方案。

过程

1. 在“仪表板”窗口的导航窗格中，单击**操作 > 创建解决方案**。
2. 在“创建解决方案”窗口中选择下列选项之一。

选项	描述
已安装的解决方案	根据 IBM Security Directory Integrator Server 上已安装的解决方案创建解决方案。
模板	基于作为模板上载的现有解决方案创建解决方案。
缺省	创建 EasyETL 解决方案，它有一个带两个连接器的组装流水线。

3. 在**解决方案名称**字段中输入解决方案的名称。
4. 单击**确定**。

解决方案配置

仪表板包含各种编辑器，用于配置数据集成解决方案的组件。

在仪表板中，可以在部署前根据您的需求以适当设置修改所选解决方案。为了便于快速配置，可以限制在配置解决方案时可见的信息量。还可以添加新的配置页面，以加入特定于解决方案的属性。

在“仪表板”窗口的导航面板上仅显示已安装的数据集成解决方案。每个解决方案包含组装流水线和关联连接器等组件。这些组件在解决方案编辑器中以树结构显示。在解决方案编辑器中，您可以使用下列编辑器安装、创建、配置、部署和监视解决方案:

- 组装流水线编辑器 – 此编辑器用于在组装流水线中创建和配置调度，以及运行组装流水线并在日志查看器中显示其输出。

- 连接器编辑器 – 此编辑器用于配置组装流水线的连接器。
- EasyETL 编辑器 – 此编辑器用于配置包含一个组装流水线及两个连接器的 EasyETL 解决方案。
- 监视器编辑器 – 此编辑器用于监视解决方案当前及以往性能的信息。

解决方案编辑器有以下菜单选项:

选项	描述
编辑解决方案界面	使用此选项可选择解决方案的组件，可以直观地编辑这些组件。
新建组装流水线	使用此选项可创建 EasyETL 解决方案并对其添加所选解决方案。
重命名组装流水线	使用此选项可重命名所选组装流水线。
删除组装流水线	使用此按钮可删除组装流水线。
创建 RunReport	使用此选项可创建 RunReport 以发送关于组装流水线执行状态的电子邮件。

添加解决方案描述

使用仪表板的解决方案编辑器可添加关于数据集成解决方案的重要信息。

过程

1. 在“仪表板”窗口的导航窗格中，选择解决方案。
2. 单击**解决方案 > 配置**。
3. 在解决方案编辑器中，选择**解决方案描述**，然后单击**编辑描述**。
4. 在文本区域中输入描述。
5. 单击**关闭**。
6. 单击**保存**。

配置 AssemblyLine 调度

使用组装流水线编辑器可创建或配置仪表板调度程序，以在指定时间运行数据集成解决方案的组装流水线。

创建调度

过程

1. 从树结构选择组装流水线。
2. 在组装流水线编辑器中，单击**调度**选项卡。
3. 单击**创建调度**。
4. 定义下列设置:

表 15. 调度选项

设置	选项	描述
调度	**解决方案启动时自动启动**	使用此选项可根据选择的执行选项运行组装流水线。
	如果已在运行则不启动	
	如果组装流水线故障则终止调度	
月	**每月**	使用此按钮可选择调度执行月份。
	选择月份	
天	**每天**	使用此按钮可选择调度执行日期。
	工作日	
	选择日期	
小时/分钟/秒	**小时**	使用此按钮可选择调度执行时间。
	分钟	
	秒	
共享日志记录		使用此选项可指定是否在调度程序和组装流水线之间共享日志记录。

5. 单击**保存**。

删除调度

过程

1. 从树结构选择组装流水线。
2. 在组装流水线编辑器中，单击**调度**选项卡。
3. 单击**删除调度**。
4. 单击**保存**。

运行和停止仪表板调度程序

过程

1. 从树结构选择组装流水线。
2. 在组装流水线编辑器中，单击**运行**选项卡。
3. 要运行组装流水线，单击**运行**。输出会显示在日志查看器中。
4. 要停止组装流水线，单击**停止**。

配置连接器

使用连接器编辑器可配置连接器详细信息（例如属性的映射信息和连接详细信息），以适应您的配置需求。

修改连接详细信息

过程

1. 从解决方案的树结构选择连接器。
2. 单击**连接器**选项卡。
3. 要仅查看重要字段，请单击**简略**。

4. 要更改连接器类型，单击**选择组件**然后从**选择连接器**列表选择连接器。
5. 根据需求，修改连接设置。
6. 要测试与数据源的连接，单击**测试连接**。
7. 单击**保存**。

修改属性映射

过程

1. 从解决方案的树结构选择连接器。
2. 单击**属性映射**选项卡。
3. 要添加属性:
   a. 单击**添加**。
   b. 从列表选择工作属性或在文本字段中输入新名称。
   c. 单击**确定**。
4. 要读取连接器数据并显示在属性映射中，单击**读取后续**。
5. 要关闭与数据源的连接，单击**关闭连接**。
6. 要修改所选属性，单击**操作**。
   a. 要编辑所选属性的分配，单击**编辑属性**并修改下列分配类型:
      - **简单分配** – 可以从源属性列表分配属性。
      - **脚本分配** – 可以在文本区域中编写用于属性的脚本。
   b. 单击**关闭**。
   c. 要映射所选属性，单击**映射属性**。
   d. 要除去所选属性的映射，单击**取消映射属性**。
7. 单击**保存**。

仪表板 EasyETL

仪表板 EasyETL 功能部件可用于创建和配置简单的数据集成解决方案。

仪表板 EasyETL 提供了用户界面，用于构建和配置带有一个源连接器和一个目标连接器的简单组装流水线。可以将 EasyETL 解决方案:

- 在仪表板调度程序中调度
- 另存为模板
- 包含在 RunReport 中
- 部署并在仪表板解决方案监视器中监视

配置 EasyETL 解决方案

使用 EasyETL 编辑器可配置 EasyETL 解决方案和向其添加调度。

过程

1. 在“仪表板”窗口的导航窗格中，选择 EasyETL 解决方案。
2. 单击**解决方案 > 配置**。

   或者，

a. 右键单击所选解决方案。
b. 在菜单中，单击**配置**。

3. 在解决方案编辑器中，选择 EasyETL 解决方案。
4. 向解决方案添加描述。请参阅第 207 页的『添加解决方案描述』主题以获取更多详细信息。
5. 在 EasyETL 编辑器中，选择解决方案并单击**配置**。
6. 从**连接器**列表为源连接器和目标连接器选择组件。
7. 在表单中指定所选连接器的配置详细信息。
8. 从**解析器**列表选择解析器。

   注: **解析器**列表仅在所选连接器需要或可以使用解析器时才处于活动状态。

   要建立连接并浏览数据:

   a. 要建立连接并查看连接器的前 25 条记录，单击**连接**。
   b. 要查看接下来的 25 条记录，单击**下一页**。
   c. 要切换数据记录视图以一次显示一条记录，单击**切换视图**。
   d. 要在切换后的视图中查看接下来的记录，单击**下一页**。
   e. 要在表中仅查看所选属性:
      1) 单击菜单栏上的**配置选项**图标。
      2) 在“配置选项”窗口中，选择属性。
      3) 要更改属性的顺序，单击向上或向下箭头。
      4) 单击**确定**。

9. 单击**后退**返回到 EasyETL 编辑器。
10. 要查看所配置连接器的属性，单击**发现**。
11. 要将所选源属性映射到目标属性，单击**映射**。 会根据您的源属性选择创建下列映射类型:
    - 如果在源视图中和目标视图中各选择了一个属性，会在两个属性之间创建一对一映射。
    - 如果在源视图中选择了多个属性，在目标视图中未选择任何属性，会为每个属性创建一对一映射。如果目标视图中无属性，会创建属性。
    - 如果在源视图中选择了多个属性，在目标视图中选择了一个属性，请执行下列任务之一:
      – 单击**复制**将源属性复制到与其对等的目标属性。
      – 单击**合并**将源中的所有值合并到目标中的一个属性。
      – 单击**连接**将源属性的值连接成一个用于目标属性的值。
12. 单击**保存**。

服务器配置

使用“服务器信息”窗口可查看和修改服务器配置。

您可以在“服务器信息”窗口中查看下列服务器属性和配置行为:

- 指定要创建和维护的最大日志文件数
- 启动和停止墓碑管理器
- 定义用于认证访问仪表板应用程序的用户的 LDAP 服务器设置
- 查看服务器信息，例如 IBM Security Directory Integrator 版本、构建日期、主机名、IP 地址、服务器引导时间、操作系统名称和服务器标识
- 查看安装在 IBM Security Directory Integrator Server 上的连接器和解析器
- 查看 IBM Security Directory Integrator Server 正在使用的各种系统存储的内容

在“仪表板”窗口左下角，可以查看应用程序的版本信息。还可以监视下列度量以获取应用程序执行情况的快照:

- 内存使用情况饼图
- 用于服务器进程的活动线程计数

配置日志设置

使用“日志和墓碑”选项卡可启用或禁用缺省记录器并指定要保存的日志文件的最大数量。

过程

1. 在“仪表板”窗口，单击**操作 > 显示服务器详细信息**或单击**详细**。
2. 单击**日志和墓碑**选项卡。
3. 定义下列设置:

选项	描述
缺省记录器	启用缺省记录器 **注**: 如果要获得组装流水线的单独日志文件，请使用缺省记录器，它会附加到每个运行的组装流水线。
最大日志文件数	指定记录器可以保存的最大日志文件数。

4. 单击**更新**。

配置墓碑

使用“日志和墓碑”选项卡可启动仪表板墓碑管理器，它会创建墓碑记录。

关于此任务

仪表板的墓碑管理器在每个组装流水线终止时为其创建墓碑记录。墓碑记录包含一些统计信息，例如迭代器读取的条目数、已跳过的记录数和已更新的记录数）。这些统计信息可用于以图形方式显示不同时间的工作负载。

过程

1. 在“仪表板”窗口，单击**操作 > 显示服务器详细信息**或单击**详细**。
2. 要生成墓碑记录，单击**启动墓碑管理器**。

   注: 要停止墓碑管理器，请重新启动 IBM Security Directory Integrator Server。

配置仪表板安全设置

使用“安全和连接”选项卡可配置 LDAP 服务器，用于认证本地和远程连接的用户。

过程

1. 在“仪表板”窗口，单击**操作 > 显示服务器详细信息**或单击**详细**。
2. 单击**安全和连接**选项卡。
3. 定义下列设置:

选项	描述
本地	指定来自本地主机的连接
远程	指定来自非本地主机的连接
LDAP 主机名	指定 LDAP 服务器的名称
LDAP 搜索条件	指定用于定位用户的 LDAP 搜索条件名称
LDAP 组 DN	指定用于检查用户成员资格的 LDAP 组 DN 名称

4. 单击**更新**。

查看已安装组件

使用“已安装组件”选项卡可查看 IBM Security Directory Integrator Server 上安装的连接器和解析器等组件。

过程

1. 在“仪表板”窗口，单击**操作 > 显示服务器详细信息**或单击**详细**。
2. 要查看组件的列表，单击**已安装组件**选项卡。

查看系统存储数据

使用“服务器存储”选项卡可查 IBM Security Directory Integrator Server 正在使用的各种系统存储的内容。

过程

1. 在“仪表板”窗口，单击**操作 > 显示服务器详细信息**或单击**详细**。
2. 要查看服务器存储的列表，单击**服务器存储**选项卡。

仪表板 RunReport

使用仪表板 RunReport 功能部件可创建组装流水线执行历史记录的自动电子邮件报告。

可以创建根据设定的调度运行的组装流水线，以生成受监视组装流水线的自动电子邮件报告。 RunReport AssemblyLine 使用 IBM Security Directory Integrator 墓碑数据库可确定上次执行 RunReport 以来是否运行过 AssemblyLine。

电子邮件报告包含关于执行历史记录的信息，显示受监视的组装流水线是否成功运行。此报告会定期通过邮件发送给指定接收方。

创建 RunReport

使用“创建 RunReport”窗口可创建 RunReport 组装流水线。此组装流水线用于生成自动电子邮件报告。这些电子邮件报告包含受监视的组装流水线的执行历史记录信息，定期发送给指定的接收方。

创建和调度 RunReport

过程

1. 在解决方案编辑器中，单击**操作 > 创建 RunReport**。
2. 在“创建 RunReport”窗口中，在**名称**字段输入 RunReport 组装流水线的名称。
3. 单击**确定**。
4. 在组装流水线编辑器中，定义下列设置:

表 16. RunReport 调度选项

设置	选项	描述
调度	**解决方案启动时自动启动**	根据选择的执行选项运行组装流水线
	如果已在运行则不启动	
	如果组装流水线故障则终止调度	
月	**每月**	指定调度执行月份
	选择的月份	
天	**每天**	指定调度执行日期
	工作日	
	选择的日期	
小时/分钟/秒	**小时**	指定调度执行时间
	分钟	
	秒	
主题（成功）		指定受监视的组装流水线执行成功的电子邮件主题行 **注**: 生成的报告根据自上次生成报告以来受监视的组装流水线是否运行成功而使用该主题行。
主题（失败）		指定受监视的组装流水线执行失败的电子邮件主题行
发送方地址		指定发送方的电子邮件地址
接收方地址		指定接收方的电子邮件地址 **注**: 如有多个电子邮件地址，请指定逗号或分号定界的列表。

表 16. RunReport 调度选项 (续)

设置	选项	描述
SMTP 主机		接受 SMTP 连接并将邮件传输给接收方的 SMTP 主机参数
监视组装流水线		以逗号分隔的要监视的组装流水线名称列表
共享日志记录		指定是否在调度程序和组装流水线之间共享日志记录

5. 单击**保存**。

删除调度

过程

1. 从树结构选择 RunReport 组装流水线。
2. 在组装流水线编辑器中，单击**调度**选项卡。
3. 单击**删除调度**。
4. 单击**保存**。

运行和停止 RunReport 调度程序

过程

1. 从树结构选择 RunReport 组装流水线。
2. 在组装流水线编辑器中，单击**运行**选项卡。
3. 要运行组装流水线，单击**运行**。输出会显示在日志查看器中。
4. 要停止组装流水线，单击**停止**。

配置和浏览连接器数据

使用解决方案编辑器中的“浏览数据”页面可配置连接器，浏览连接器的数据源，或创建数据集成解决方案。

过程

1. 在“仪表板”窗口的导航窗格中，单击**浏览数据**。
2. 在“浏览数据”窗口中，从**连接器**列表选择要配置的连接器并浏览其内容。
3. 从**解析器**列表选择解析器。

   注: **解析器**列表仅在所选连接器需要或可以使用解析器时才处于活动状态。
4. 在窗口右侧的表单中配置连接器详细信息。
5. 要为所选连接器创建解决方案，单击**创建集成**。
6. 要建立连接并查看连接器的前 25 条记录，单击**连接**。
7. 要查看接下来的 25 条记录，单击**下一页**。
8. 要切换数据记录视图以一次显示一条记录，单击**切换视图**。
9. 要在切换后的视图中查看接下来的记录，单击**下一页**。
10. 要在表中仅查看所选属性:
    a. 在“浏览数据”窗口中单击**配置选项**图标。

b. 在“配置选项”窗口中，选择属性。
c. 要更改属性的顺序，单击向上或向下箭头。
d. 单击**确定**。

解决方案监视器

使用仪表板监视器编辑器可跟踪组装流水线执行进度并查看解决方案的执行历史记录。

可以在监视器编辑器中查看所选解决方案的组装流水线，该编辑器显示组装流水线的当前及以往性能信息。可以监视下列活动:

- 组装流水线的实时状态监视，包括:
  - 开始执行组装流水线的时间
  - 结束执行组装流水线的时间
  - 下次运行组装流水线的调度时间
  - 在组装流水线中处理的数据对象数
- 组装流水线的执行历史记录，以图形方式显示不同时间的工作负载
- 记录每次执行组装流水线的结果和问题的日志文件
- 解决方案中所有组装流水线的墓碑记录
- 服务器日志文件 (`ibmdi.log`)

启动和停止组装流水线

使用仪表板监视器编辑器可以启动和停止组装流水线。

过程

1. 在仪表板中，选择已安装的解决方案。
2. 单击**解决方案 > 监视**。

   或者，

   a. 右键单击所选解决方案。
   b. 在菜单中，单击**监视**。
3. 在监视器编辑器中，选择组装流水线以执行下列操作:
   - 要启动组装流水线，单击**启动**。
   - 要停止组装流水线，单击**停止**。

   执行详细信息会显示在编辑器中。带组装流水线名称的绿色图标表示组装流水线处于活动状态。

查看组装流水线执行历史记录

使用仪表板监视器编辑器可以图形形式查看组装流水线执行历史记录。

过程

1. 在编辑器编辑器中，选择组装流水线。
2. 单击**历史记录**或双击所选组装流水线。 组装流水线执行详细信息会在图形视图中显示为不同时间的工作负载。

3. 选择以下任一计数器以绘图:
   - 获取
   - 错误
   - 添加
   - 删除
   - 查找
   - 修改
4. 单击图形上的节点可查看特定运行的日志文件。还可以工具提示的方式查看该运行的执行统计信息。

查看墓碑记录

使用仪表板监视器编辑器中的“墓碑”选项卡可查看在组装流水线执行完成时创建的墓碑记录。

过程

1. 转到仪表板的监视器编辑器。
2. 要查看记录的墓碑，单击**墓碑**选项卡。

   注: 只有在墓碑管理器运行时才能查看墓碑记录。有关更多信息，请参阅第 211 页的『配置墓碑』主题。

查看日志文件

使用仪表板监视器编辑器中的“日志文件”选项卡可查看为每次组装流水线执行创建的日志文件。日志文件用于快速方便地分析问题和调试问题。

过程

1. 在监视器编辑器中，单击**日志文件**选项卡。所有组装流水线执行的日志文件会以树结构显示。
2. 要查看日志文件的内容，从树结构选择日志文件并双击。
3. 要在日志内容中搜索特定信息，在**搜索**字段中输入要搜索的文本。
4. 选择**包括源**复选框可包括消息源或记录消息的组件。
5. 单击**选项**可设置下列详细信息:

选项	描述
页面大小	
消息类型	在下列消息类型中选择任何要包括在日志文件中的类型: • 参考 • 警告 • 错误 • 调试

选项	描述
显示选项	选择用于在日志文件中显示日期、时间、消息源或行号的选项: • 显示日期 • 显示时间 • 包括源 • 显示行号

声明

本信息是为在美国国内供应的产品和服务而编写的。IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您所在区域当前可获得的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区:

International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。

某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本出版物的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

所有 IBM 的价格均是 IBM 当前的建议零售价，可随时更改而不另行通知。经销商的价格可与此不同。

本信息仅用于规划的目的。在所描述的产品上市之前，此处的信息可随时更改。

这些信息包含日常业务操作中使用的数据和报告示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称均是虚构的，如与实际的商业企业使用的名称和地址有任何相似之处，纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。用户如果是为了按照 IBM 应用程序编程接口开发、使用、经销或分发应用程序，则可以任何形式复制、修改和分发这些样本程序，而无须向 IBM 付费。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

© (贵公司名称) (年份).此部分代码是根据 IBM 公司的样本程序衍生出来的。 © Copyright IBM Corp. （输入年份）.All rights reserved.

如果您正以软拷贝格式查看本信息，图片和彩色图例可能无法显示。

Trademarks

IBM、IBM 徽标和 ibm.com® 是 International Business Machines Corp.，在全球许多管辖区域的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。IBM 商标的最新列表可从 Web 站点 www.ibm.com/legal/copytrade.shtml 上的“版权和商标信息”部分获取。

Adobe、Acrobat、PostScript 和所有基于 Adobe 的商标是 Adobe Systems Incorporated 在美国和/或其他国家或地区的注册商标或商标。

IT Infrastructure Library 是 Central Computer and Telecommunications Agency 的注册商标，该企业现已成为 Office of Government Commerce 的一部分。

Intel、Intel徽标、Intel Inside、Intel Inside 徽标、Intel Centrino、Intel Centrino 徽标、Celeron、Intel Xeon、Intel SpeedStep、Itanium 和 Pentium 是 Intel Corporation 或其子公司在美国和其他国家或地区的商标或注册商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的商标。

Microsoft、Windows、Windows NT 和 Windows 徽标是 Microsoft Corporation 在美国和/或其他国家或地区的商标。

ITIL 是英国政府商务部的注册商标和欧盟注册商标，且已在美国专利和商标局注册。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

Java 和所有基于 Java 的商标和徽标是 Oracle 和/或其子公司的商标或注册商标。

Cell Broadband Engine 是 Sony Computer Entertainment, Inc. 在美国和/或其他国家或地区的商标，并已授权使用。

Linear Tape-Open、LTO、LTO 徽标、Ultrium 和 Ultrium 徽标是 HP、IBM Corp. 和 Quantum 在美国和其他国家或地区的商标。

索引

P

S

T

U

W

[特别字符]

Printed in China

S151-1410-03