

# 目录

Introduction	1.1
01-高级架构	1.2
02-类型系统	1.3
03-基本搜索	1.4
04-高级搜索 <del>X</del>	1.5
05-术语表	1.6
06-安全	1.7
07-认证	1.8
08-Atlas授权模型	1.9
8.1 配置Atlas Simple Authorizer	1.9.1
8.2 配置Atlas Ranger Authorizer	1.9.2
09-分类传播	1.10
10-配置Atlas属性	1.11
11-通知	1.12
12-Hook & Bridge	1.13
12.1 HBase Hook & Bridge	1.13.1
12.2 Hive Hook&Bridge <del>X</del>	1.13.2
12.3 Sqoop Hook <del>X</del>	1.13.3
12.4 Storm Hook <del>X</del>	1.13.4
12.5 Kafka Bridge <del>X</del>	1.13.5
13-容错和高可用性选项 <del>X</del>	1.14
14-从Apache Atlas 0.8迁移 <del>X</del>	1.15

- Hadoop的数据治理和元数据框架
  - 1. 概述
  - 2. 特性

# Hadoop的数据治理和元数据框架

笔者近期在和团队的小伙伴进行元数据管理方向的探索，此篇文档翻译自[Atlas官方文档](#)。希望对大家有帮助，欢迎大家交流！

## 1. 概述

Atlas 是一个可伸缩和可扩展的核心基础治理服务集合，使企业能够有效地和高效地满足 Hadoop 中的合规性要求，并允许与整个企业数据生态系统的集成。

Apache Atlas为组织提供开放式元数据管理和治理功能，用以构建其数据资产目录，对这些资产进行分类和管理，并为数据科学家，数据分析师和数据治理团队提供围绕这些数据资产的协作功能。

## 2. 特性

### 元数据类型 & 实例

- 各种Hadoop和非Hadoop元数据的预定义类型
- 能够为要管理的元数据定义新类型
- 类型可以具有原始属性，复杂属性，对象引用；可以继承其他类型
- 类型(type)实例（称为实体entities）捕获元数据对象详细信息及其关系
- 可以更轻松地进行集成用于处理类型和实例的REST API

### 分类

- 能够动态创建分类 - 如PII, EXPIRES\_ON, DATA\_QUALITY, SENSITIVE。
- 分类可以包含属性 - 例如EXPIRES\_ON分类中的expiry\_date属性。
- 实体(entities)可以与多个分类(classifications)相关联，从而实现更轻松的发现和安全实施。
- 通过血缘传播分类 - 自动确保分类在进行各种处理时遵循数据。

### 血缘

- 直观的UI，用于在数据流转时，通过各种流程时查看数据。
- 用于访问和更新血缘的REST API。

### 搜索/发现

- 直观的UI，按类型(type)，分类(classification)，属性值(attribute)或自由文本搜索实体。
- 丰富的REST API，可按复杂条件进行搜索。
- SQL搜索实体的查询语言 - 域特定语言 (DSL) 。

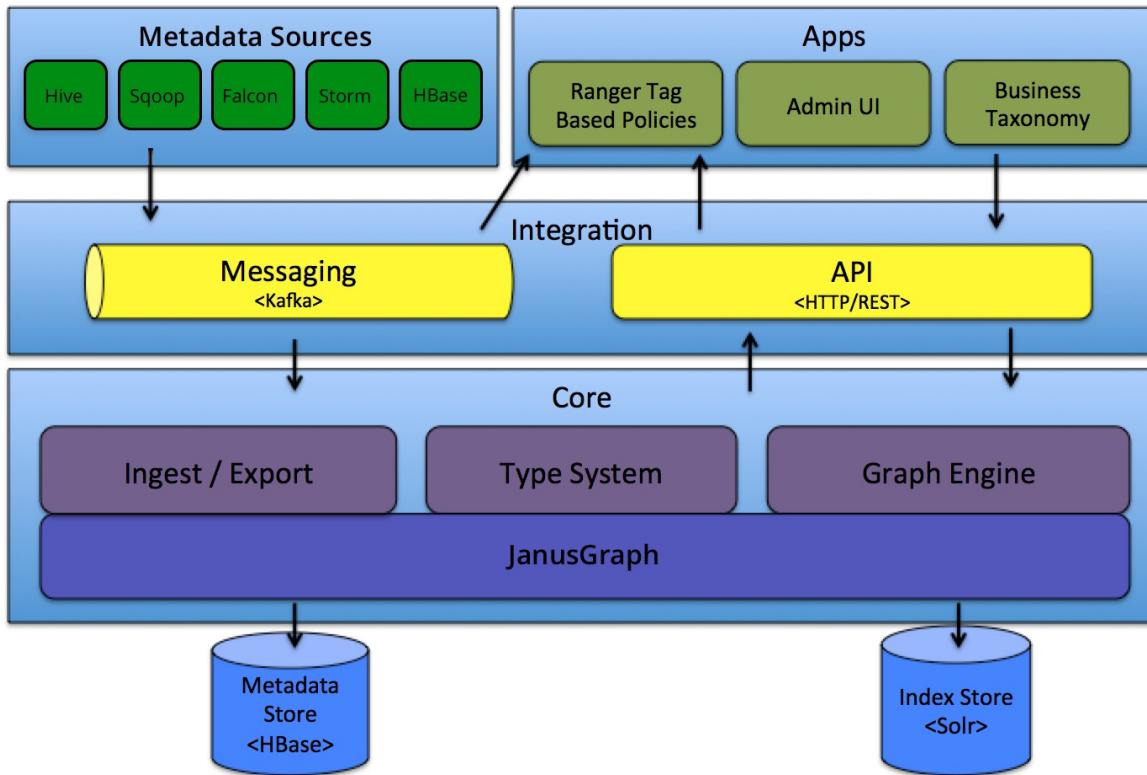
### 安全和数据屏蔽

- 用于元数据访问的细粒度安全性，实现对实体实例的访问控制以及添加/更新/删除分类等操作。
- 与Apache Ranger集成可根据与Apache Atlas中的实体相关的分类对数据访问进行授权/数据屏蔽。例如：

- 谁可以访问分类为PII, SENSITIVE的数据。
- 客户服务用户只能看到分类为NATIONAL\_ID的列的最后4位数字。

- 01.高级架构

## 01.高级架构



Atlas的组件可以分为以下主要类别：

### Core层

Atlas核心包含以下组件：

**类型(Type)系统:** Atlas允许用户为他们想要管理的元数据对象定义模型。该模型由称为“类型”的定义组成。称为“实体”的“类型”实例表示受管理的实际元数据对象。 Type System是一个允许用户定义和管理类型和实体的组件。开箱即用的Atlas管理的所有元数据对象（例如Hive表）都使用类型建模并表示为实体。要在Atlas中存储新类型的元数据，需要了解类型系统组件的概念。

需要注意的一个关键点是Atlas中建模的一般特性允许数据管理员和集成商定义技术元数据和业务元数据。也可以使用Atlas的功能定义两者之间的丰富关系。

**图形引擎:** Atlas在内部使用Graph模型持久保存它管理的元数据对象。这种方法提供了很大的灵活性，可以有效地处理元数据对象之间的丰富关系。图形引擎组件负责在Atlas类型系统的类型和实体之间进行转换，以及底层图形持久性模型。除了管理图形对象之外，图形引擎还为元数据对象创建适当的索引，以便可以有效地搜索它们。Atlas使用JanusGraph存储元数据对象。

**采集/导出:** 采集组件允许将元数据添加到Atlas。同样，“导出”组件将Atlas检测到的元数据更改公开为事件。消费者可以使用这些更改事件来实时响应元数据的变更。

### Integration层

在Atlas中，用户可以使用以下的两种方式管理元数据：

API: Atlas的所有功能都通过REST API向最终用户暴露，该API允许创建，更新和删除类型和实体。它也是查询和发现Atlas管理的类型和实体的主要机制。

Messaging: 除了API之外，用户还可以选择使用基于Kafka的消息传递接口与Atlas集成。这对于将元数据对象传递到Atlas以及使用Atlas使用可以构建应用程序的元数据更改事件都很有用。如果希望使用与Atlas更松散耦合的集成来实现更好的可伸缩性，可靠性等，则消息传递接口特别有用。Atlas使用Apache Kafka作为通知服务器，用于钩子和元数据通知事件的下游消费者之间的通信。事件由钩子和Atlas写入不同的Kafka主题。

## Metadata sources

Atlas支持开箱即用的多种元数据源集成。未来还将增加更多集成。目前，Atlas支持从以下来源提取和管理元数据：

- HBase
- Hive
- Sqoop
- Storm
- Kafka

集成意味着两件事：Atlas定义的元数据模型用于表示这些组件的对象。Atlas提供了从这些组件中摄取元数据对象的组件（在某些情况下实时或以批处理模式）。

## Applications

Atlas管理的元数据被各种应用程序使用，以满足许多治理需求。

Atlas Admin UI: 该组件是一个基于Web的应用程序，允许数据管理员和科学家发现和注释元数据。这里最重要的是搜索界面和类似SQL的查询语言，可用于查询Atlas管理的元数据类型和对象。Admin UI使用Atlas的REST API来构建其功能。

Tag Based Policies: Apache Ranger是Hadoop生态系统的高级安全管理解决方案，可与各种Hadoop组件进行广泛集成。通过与Atlas集成，Ranger允许安全管理员定义元数据驱动的安全策略以实现有效的治理。Ranger是Atlas通知的元数据更改事件的使用者。

- 02. 类型系统
  - 概述
  - Types(类型)介绍
  - Entities(实体)
  - Attributes(属性)
  - 系统特定类型及含义

## 02. 类型系统

### 概述

Atlas允许用户为他们想要管理的元数据对象定义模型。该模型由称为 type(类型) 的定义组成。称为 entities(实体) 的 type(类型) 实例表示受管理的实际元数据对象。Type System是一个允许用户定义和管理类型和实体的组件。开箱即用的Atlas管理的所有元数据对象（例如Hive表）都使用类型建模并表示为实体。要在Atlas中存储新类型的元数据，需要了解类型系统组件的概念。

### Types(类型)介绍

Atlas中的 Type 是对特定类型的元数据对象如何存储和访问的定义。Type表示定义元数据对象属性的一个或一组属性。具有开发基础的用户能了解到，类型就相当于面向对象编程语言的“Class”定义或关系数据库的“table schema”。

使用Atlas的类型的其中一个示例是Hive表。Hive表定义了以下属性：

```
Name:          hive_table
TypeCategory: Entity
SuperTypes:   DataSet
Attributes:
  name:        string
  db:          hive_db
  owner:       string
  createTime:  date
  lastAccessTime: date
  comment:     string
  retention:   int
  sd:          hive_storagedesc
  partitionKeys: array<hive_column>
  aliases:     array<string>
  columns:    array<hive_column>
  parameters: map<string, string>
  viewOriginalText: string
  viewExpandedText: string
  tableType:   string
  temporary:   boolean
```

从上面的例子中可以注意到以下几点：

- Atlas中的类型(Type)由 name 唯一标识
- Type具有元类型。Atlas中有以下元类型：
  - 原始元类型(Primitive metatypes): boolean, byte, short, int, long, float, double, biginteger, bigdecimal, string, date
  - 枚举元型(Enum metatypes)
  - 集合元类型(Collection metatypes): array, map
  - 复合元类型(Composite metatypes): Entity, Struct, Classification, Relationship

- 实体(Entity)和分类(Classification)类型可以从其他类型继承，称为“超类型/父类型”(supertype)，它包括在超类型中定义的属性。这允许建模者在一组相关类型等中定义公共属性。类似于面向对象语言如何为类定义父类。Atlas中的类型也可以从多个超类型扩展。
  - 在此示例中，每个配置单元表都从称为 `DataSet` 的预定义超类型扩展。稍后将提供有关此预定义类型的更多详细信息。
- 具有元类型 `Entity`，`Struct`，`Classification` 或 `Relationship` 的类型可以具有属性的集合。每个属性都有一个名称（例如：`name`）和一些其他相关属性。可以使用表达式 `type_name.attribute_name` 引用属性。值得注意的是，属性本身是使用Atlas元类型定义的。
  - 在此示例中，`hive_table.name` 是 `String`，`hive_table_aliases` 是一个字符串数组，`hive_table.db` 是指一个名为 `hive_db` 的类型的实例，依此类推。
- 属性中的类型引用（如 `hive_table.db`）特别有趣，使用这样的属性，我们可以定义Atlas中定义的两种类型之间的任意关系，从而构建丰富的模型。此外，还可以将引用列表收集为属性类型（例如，`hive_table.columns`，表示从 `hive_table` 到 `hive_column` 类型的引用列表）

## Entities(实体)

Atlas中的 `entity` 是 `type` 的特定值或实例，因此表示现实世界中的特定元数据对象。用我们对面向对象编程语言的类比，实例(`instance`)是某个类(`Class`)的对象(`Object`)。

实体的其中一个示例就是Hive表。Hive在'default'数据库中有一个名为'customers'的表。该表是 `hive_table` 类型的Atlas中的“实体”。由于是实体类型的实例，它将具有作为Hive表'type'的一部分的每个属性的值，例如：

```
guid: "9ba387dd-fa76-429c-b791-ffc338d3c91f"
typeName: "hive_table"
status: "ACTIVE"
values:
  name: "customers"
  db: { "guid": "b42c6cfcc1e7-42fd-a9e6-890e0adf33bc", "typeName": "hive_db" }
  owner: "admin"
  createTime: 1490761686029
  updateTime: 1516298102877
  comment: null
  retention: 0
  sd: { "guid": "ff58025f-6854-4195-9f75-3a3058dd8dcf", "typeName": "hive_storagedesc" }
  partitionKeys: null
  aliases: null
  columns: [ { "guid": "65e2204f-6a23-4130-934a-9679af6a211f", "typeName": "hive_column" }, { "guid": "d726de70-faca-46fb-9c99-cf04f6b579a6", "typeName": "hive_column" }, ... ]
  parameters: { "transient_lastDdlTime": "1466403208" }
  viewOriginalText: null
  viewExpandedText: null
  tableType: "MANAGED_TABLE"
  temporary: false
```

从上面的例子中可以注意到以下几点：

- 实体类型的每个实例都由唯一标识符GUID标识。此GUID由Atlas服务器在定义对象时生成，并在实体的整个生命周期内保持不变。在任何时间点，都可以使用其GUID访问此特定实体。
  - 在此示例中，默认数据库中的“customers”表由GUID“9ba387dd-fa76-429c-b791-ffc338d3c91f”唯一标识。
- 实体具有给定类型，并且类型的名称随实体定义一起提供。
  - 在此示例中，'customers'表是'hive\_table'类型。
- 该实体的值是 `hive_table` 类型定义中定义的属性的所有属性名称及其值的映射。属性值将根据属性的数据类型。实体类型属性将具有 `AtlasObjectId` 类型的值

有了实体的这个设计，我们现在可以看到Entity和Struct元类型之间的区别。实体(Entity)和结构(Entity)都构成其他类型的属性。但是，实体类型的实例具有标识(具有GUID值)，并且可以从其他实体引用（例如，从hive\_table实体引用hive\_db实体）。Struct类型的实例没有自己的标识。Struct类型的值是在实体本身内“嵌入”的属性集合。

## Attributes(属性)

我们已经看到，属性(attributes)是在实体(Entity)，结构(Struct)，分类(Classification)和关系(Relationship)等元类型中定义的。但我们将属性列举为具有名称和元类型值。然而，Atlas中的attributes具有一些properties，这些properties定义了与类型系统相关的更多概念。

attributes具有以下properties：

```
name: string,
typeName: string,
isOptional: boolean,
isIndexable: boolean,
isUnique: boolean,
cardinality: enum
```

上述属性具有以下含义：

- `name`：属性的名称
- `typeName`：属性的元类型名称 (native, collection, composite))
- `isComposite`：
  - 该标志表示建模的一个方面。如果将属性定义为复合(composite)，则意味着它不能具有独立于其所包含的实体的生命周期。这个概念的一个很好的示例是构成hive表的一部分的列集。由于列在hive表外部没有意义，因此它们被定义为复合属性。
  - 必须在Atlas中创建复合属性及其包含的实体。即，必须与hive表一起创建配置单元列。
- `isIndexable`
  - 标志指示是否应该对此属性建立索引，以便可以使用属性值作为谓词来执行查找，并且可以有效地执行查找。
- `isUnique`
  - 同样与索引相关。如果指定为唯一，则表示在JanusGraph中为此属性创建了一个特殊索引，允许基于相等的查找。
  - 具有该标志的真值的任何属性都被视为主键，以将该实体与其他实体区分开。因此，应该注意确保此属性确实在现实世界中为唯一属性建模。
    - 对于例如考虑hive\_table的name属性。在单独的情况下，名称不是hive\_table的唯一属性，因为具有相同名称的表可以存在于多个数据库中。如果Atlas在多个集群中存储hive表的元数据，那么即使是一对（数据库名称，表名）也不是唯一的。在物理世界中，只有集群位置，数据库名称和表名称才能被视为唯一。
- `multiplicity`：标示该属性是必选(required)，可选(optional)的还是可以是多值的(multi-valued)。如果实体的属性值定义与类型定义中的多重性声明不匹配，则这将违反约束，并且实体添加将失败。因此，该字段可用于定义元数据信息的一些约束。

根据上面的内容，让我们展开下面的hive表的一个attributes的属性定义。让我们看一下名为'db'的属性，它表示hive表所属的数据库：

```
db:
  "name": "db",
  "typeName": "hive_db",
  "isOptional": false,
  "isIndexable": true,
  "isUnique": false,
  "cardinality": "SINGLE"
```

请注意“`isOptional = true`”约束 - 如果没有db引用，则无法创建表实体。

```

columns:
  "name": "columns",
  "typeName": "array<hive_column>",
  "isOptional": optional,
  "isIndexable": true,
  "isUnique": false,
  "constraints": [ { "type": "ownedRef" } ]

```

请注意列的“ownedRef”约束。通过这样，我们指出定义的列实体应始终绑定到它们所定义的表实体。

通过此描述和示例，您将能够意识到属性定义可用于影响Atlas系统强制执行的特定建模行为（约束，索引等）。

## 系统特定类型及含义

Atlas自带了一些预定义的系统类型。我们在前面的部分中看到了一个示例（DataSet）。在本节中，我们将看到更多这些类型并了解它们的重要性。

- **Referenceable**: 该类型表示可以使用名为qualifiedName的唯一属性搜索的所有实体。
- **Asset**: 该类型扩展了Referenceable并添加了名称，描述和所有者等属性。Name是必需属性（isOptional = false），其他属性是可选的。

Referenceable和Asset的目的是为建模者提供在定义和查询自己类型的实体时强制一致性的方法。拥有这些固定的属性集允许应用程序和用户界面基于约定做出关于默认情况下它们可以期望类型的属性的假设。

- **Infrastructure**: 该类型继承自Asset，通常可用作基础结构元数据对象（如集群，主机等）的常见超类型。
- **DataSet**: 该类型继承自Referenceable。从概念上讲，它可用于表示存储数据的类型。在Atlas中，hive表，hbase\_tables等都是从DataSet扩展的类型。扩展DataSet的类型可以预期具有Schema，因为它们具有定义该数据集的属性的属性。对于例如hive\_table中的columns属性。此外，扩展DataSet的类型实体参与数据转换，Atlas可以通过血缘（图了解到转换过程）图了解到转换过程。
- **Process**: 该类型继承自Asset。从概念上讲，它可用于表示任何数据转换操作。例如，将具有原始数据的配置单元表转换为存储某些聚合的另一个配置单元表的ETL过程可以是扩展Process类型的特定类型。流程类型有两个特定属性，即输入和输出。输入和输出都是DataSet实体的数组。因此，Process类型的实例可以使用这些输入和输出来捕获DataSet的血缘如何演变。

- 03. 基本搜索

## 03. 基本搜索

基本搜索允许您使用实体的类型名称，关联的分类/标记进行查询，并且支持对实体属性以及分类/标记属性进行过滤。

可以使用以下JSON结构（SearchParameters）表示整个查询结构：

```
{
  "typeName": "hive_column",
  "excludeDeletedEntities": true,
  "classification": "PII",
  "query": "",
  "offset": 0,
  "limit": 25,
  "entityFilters": {},
  "tagFilters": {},
  "attributes": ["table", "qualifiedName"]
}
```

### 字段解释

typeName:	要查找的实体类型
excludeDeletedEntities:	搜索是否应该排除已删除的实体？（默认值：true）
classification:	仅包括具有给定分类的实体
query:	实体应具有的任何自由文本出现（通用/通配符查询可能很慢）
offset:	结果集的起始偏移量（对分页有用）
limit:	要获取的最大结果数
entityFilters:	实体属性过滤器
tagFilters:	分类属性过滤器
attributes:	要包含在搜索结果中的属性

The screenshot shows the Apache Atlas search interface. On the left, there are search filters for 'Search By Type' (set to 'hive\_column'), 'Search By Classification' (set to 'PII'), and 'Search By Term'. The main area displays a table of search results with columns: Name, Owner, Type, Type, Classifications, Table, and QualifiedName. The results are as follows:

Name	Owner	Type	Type	Classifications	Table	Qualified Name
providername	hive	hive_column	string	VENDOR_PII	prov_view	claim.prov_view.providername@cl1
emailaddress	hive	hive_column	string	PII	ww_customers	hortoniabank.ww_customers.emailaddress@cl1
ccnumber	hive	hive_column	string	PII	ww_customers	hortoniabank.ww_customers.ccnumber@cl1
nationalid	hive	hive_column	string	PII	ww_customers	hortoniabank.ww_customers.nationalid@cl1
nationalid	hive	hive_column	string	PII	us_customers	hortoniabank.us_customers.nationalid@cl1
ssn	hive	hive_column	string	FINANCE...	tax_2015	finance.tax_2015.ssn@cl1
providername	hive	hive_column	string	VENDOR...	provider_summary	claim.provider_summary.providername@cl1
ssn	hive	hive_column	string	FINANCE...	tax_2010	finance.tax_2010.ssn@cl1

可以使用AND/OR条件对多个属性进行基于属性的过滤。

### 过滤示例（适用于hive\_table属性）

- 单个属性：

```
{
  "typeName": "hive_table",
  "excludeDeletedEntities": true,
```

```

    "offset": 0,
    "limit": 25,
    "entityFilters": {
        "attributeName": "name",
        "operator": "contains",
        "attributeValue": "customers"
    },
    "attributes": [ "db", "qualifiedName" ]
}

```

The screenshot shows the Apache Atlas search interface. On the left, there are search filters for 'Search By Type' (set to 'hive\_table'), 'Search By Classification' (set to 'Select Classification'), 'Search By Term' (set to 'Search Term'), 'Search By Text' (set to 'Search by text'), and a 'Basic' search bar ('hive\_table - customers'). On the right, a modal window titled 'Attribute Filter' is open, showing an OR condition with a single filter: 'Name (string)' contains 'customers'. Below the modal, the search results table lists entities: 'us\_customers' and 'hive\_table'. The 'hive\_table' row is selected, showing its details: 'qualifiedName' is 'hortoniabank.wv\_customers@cl1'. At the bottom of the interface, there is a 'Page Limit' dropdown set to 25.

- 包含OR条件的多个属性

```

{
    "typeName": "hive_table",
    "excludeDeletedEntities": true,
    "offset": 0,
    "limit": 25,
    "entityFilters": {
        "condition": "OR",
        "criterion": [
            {
                "attributeName": "name",
                "operator": "contains",
                "attributeValue": "customers"
            },
            {
                "attributeName": "name",
                "operator": "contains",
                "attributeValue": "provider"
            }
        ],
        "attributes": [ "db", "qualifiedName" ]
}

```

The screenshot shows the Apache Atlas search interface. On the left, there's a sidebar with search filters like 'Search By Type' (hive\_table), 'Search By Classification' (Select Classification), 'Search By Term' (Search Term), and 'Search By Text' (Search by text). The main area is titled 'Attribute Filter' with the condition 'AND'. It has two rows of filters. The first row has 'Name (string)' as the attribute, 'contains' as the operator, and 'customers' as the value. The second row also has 'Name (string)' as the attribute, 'contains' as the operator, and 'provider' as the value. A tooltip for 'contains' is visible, showing options: contains, !=, begins with, ends with, is null, and is not null. At the bottom right of the dialog are 'Cancel', 'Apply', and 'Search' buttons.

- 包含AND条件的多个属性

```
{
  "typeName": "hive_table",
  "excludeDeletedEntities": true,
  "offset": 0,
  "limit": 25,
  "entityFilters": [
    {
      "condition": "AND",
      "criterion": [
        {
          "attributeName": "name",
          "operator": "contains",
          "attributeValue": "customers"
        },
        {
          "attributeName": "owner",
          "operator": "eq",
          "attributeValue": "hive"
        }
      ]
    },
    {
      "attributes": [ "db", "qualifiedName" ]
    }
}
```

This screenshot is similar to the one above, showing the Apache Atlas search interface with the 'Attribute Filter' dialog open. The filter condition is 'AND'. The first row has 'Name (string)' as the attribute, 'contains' as the operator, and 'customers' as the value. The second row has 'Owner (string)' as the attribute, 'eq' as the operator, and 'hive' as the value. A tooltip for 'eq' is visible, showing options: contains, !=, begins with, ends with, is null, and is not null. The bottom right of the dialog has 'Cancel', 'Apply', and 'Search' buttons.

支持过滤的运算符

- LT (操作符: <, lt) 与Numeric, Date属性一起使用
- GT (操作符: >, gt) 与Numeric, Date属性一起使用
- LTE (操作符: <=, lte) 与Numeric, Date属性一起使用
- GTE (操作符: >=, gte) 与Numeric, Date属性一起使用
- EQ (操作符: eq, =) 与Numeric, Date, String属性一起使用
- NEQ (操作符: neq, !=) 与Numeric, Date, String属性一起使用
- LIKE (操作符: like, LIKE) 与String属性一起使用
- STARTS\_WITH (symbols: startsWith, STARTSWITH) 与String属性一起使用
- ENDS\_WITH (symbols: endsWith, ENDSWITH) 与String属性一起使用
- CONTAINS (操作符: contains, CONTAINS) 与String属性一起使用

#### CURL示例

```
curl -sivk -g
-u <user>:<password>
-X POST
-d '{
    "typeName": "hive_table",
    "excludeDeletedEntities": true,
    "classification": "",
    "query": "",
    "offset": 0,
    "limit": 50,
    "entityFilters": {
        "condition": "AND",
        "criterion": [
            {
                "attributeName": "name",
                "operator": "contains",
                "attributeValue": "customers"
            },
            {
                "attributeName": "owner",
                "operator": "eq",
                "attributeValue": "hive"
            }
        ]
    },
    "attributes": [ "db", "qualifiedName" ]
}'
<protocol>://<atlas_host>:<atlas_port>/api/atlas/v2/search/basic
```

- 05. 术语

## 05. 术语

Atlas的术语表(Glossary)提供了一些适当的“单词”，这些“单词”能彼此进行关联和分类，以便业务用户在使用的时候，即使在不同的上下文中也能很好的理解它们。此外，这些术语也是可以映射到数据资产中的，比如：数据库，表，列等。

术语表抽象出了和数据相关的专业术语，使得用户能以他们更熟悉的方式去查找和使用数据。

### 术语表功能

- 能够使用自然语言（技术术语和/或业务术语）定义丰富的术语词汇表。
- 能够将术语在语义上相互关联。
- 能够将资产映射到术语表中。
- 能够按类别划分这些术语。这为术语增加了更多的上下文。
- 允许按层次结构排列类别，能展示更广泛和更精细的范围。
- 从元数据中独立管理术语表。

### 术语(Term)

对于企业来说术语作用的非常大的。对于有用且有意义的术语，需要围绕其用途和上下文进行分组。Apache Atlas中的术语必须具有唯一的qualifiedName，可以有相同名称的术语，但它们不能属于同一个术语表。具有相同名称的术语只能存在于不同的术语表中。

术语名称可以包含空格，下划线和短划线（作为引用单词的自然方式）但不包含“.”或“@”，因为qualifiedName的格式为：`<术语>@<术语限定名>`。限定名称可以更轻松地使用特定术语。

术语只能属于单个术语表，并且它们的生命周期也是相同的，如果删除术语表，则术语也会被删除。术语可以属于零个或多个类别，这允许将它们限定为更小或更大的上下文。

可以在Apache Atlas中为一个或多个实体分配/链接一个术语。可以使用分类（`classifications`，类似标签的作用）对术语进行分类，并将相同的分类应用于分配术语的实体。

### 类别(Category)

类别是组织术语的一种方式，以便可以丰富术语的上下文。

类别可能包含也可能不包含层次结构，即子类别层次结构。类别的qualifiedName是使用它在术语表中的分层位置导出的，例如：`<类别名称>.<父类别限定名>`。当发生任何层级更改时，此限定名称都会更新，例如：添加父类别，删除父类别或更改父类别。

### Atlas Web UI

Apache Atlas UI提供了友好的用户界面，可以使用术语表相关的功能，其中包括：

- 创建术语表，术语和类别
- 在术语之间创建各种关系：synonymns(同义词)，antonymns(反义词)，seeAlso(参考)
- 调整类别的层次结构中
- 为实体分配实体(entities)
- 使用关联术语搜索实体

与术语表相关的UI都可以在 GLOSSARY 的Tab下找到。

### Glossary tab

Apache Atlas UI提供了两种使用术语表的方法：术语(Terms)视图 和 类别(Category)视图。

(1) 术语视图(Terms)

术语视图允许用户执行以下操作：

- 创建，更新和删除术语
- 添加，删除和更新与术语关联的分类
- 添加，删除和更新术语的分类
- 在术语之间创建各种关系
- 查看与术语关联的实体

(2) 类别视图(Category)

类别视图允许用户执行以下操作：

- 创建，更新和删除类别和子类别
- 将术语与类别相关联

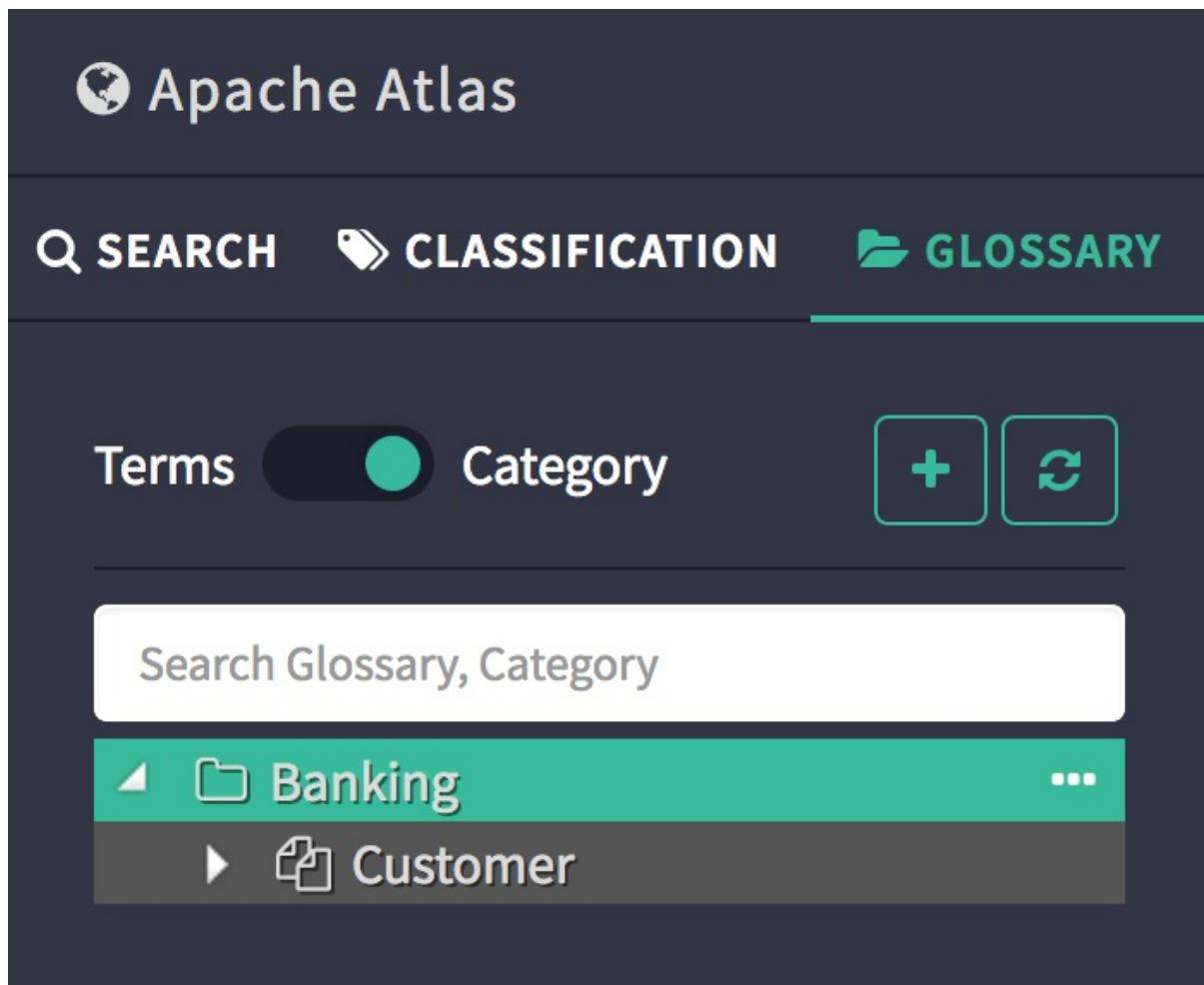
用户可以使用术语表选项卡中提供的切换在术语视图和类别视图之间切换。

The screenshot shows the Apache Atlas interface with the "GLOSSARY" tab selected. A toggle switch at the top left allows switching between "Terms" (selected) and "Category". Below the search bar, a category tree is displayed under the heading "Banking".

Search Glossary, Term

◀ **Banking** ...

- 📄 15-30 yr mortgage
- 📄 A checking account
- 📄 A savings account
- 📄 ARM loans



### 术语菜单(Terms)

- 创建一个新术语  
单击术语表名称旁边的省略号 (...) 会显示一个弹出式菜单，允许用户在术语表中创建术语或删除术语表 - 如下所示。

The screenshot shows the Apache Atlas interface with the 'GLOSSARY' tab selected. A search bar at the top contains the placeholder 'Search Glossary, Term'. Below it, a tree view shows the 'Banking' category expanded, containing terms like '15-30 yr mortgage', 'A checking account', 'A savings account', 'ARM loans', and 'Tax'. A context menu is open over the term '15-30 yr mortgage', listing options to 'Create Term' and 'Delete Glossary'.

- 删除一个术语

单击术语名称旁边的省略号 (...) 会显示一个弹出式菜单，允许用户删除该术语 - 如下所示。

The screenshot shows the Apache Atlas interface displaying detailed information for the term '15-30 yr mortgage'. The term is highlighted in green. A context menu is open over the term, with the 'Delete Term' option highlighted. Other visible details include the short and long descriptions, classifications (none shown), categories ('Loans'), and related terms.

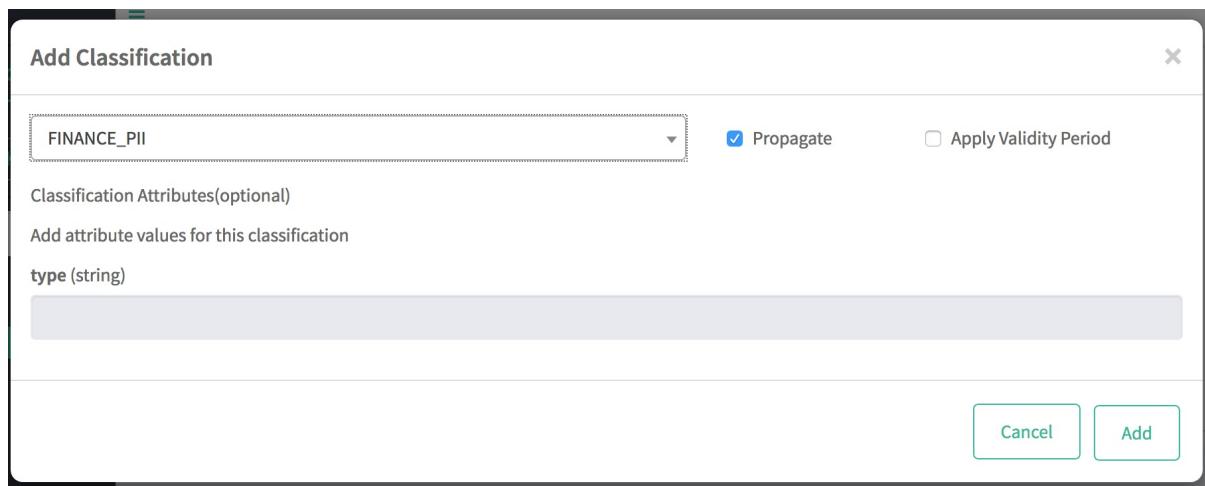
术语详情

选择术语表UI中的术语，可以查看对应术语的各种详细信息。详细信息页面下的每个选项卡提供该术语的不同详细信息。

- Entities(实体) 选项卡：显示分配给所选术语的实体
- Classifications(分类) 选项卡：显示与所选术语关联的分类
- Related terms (相关术语) 选项卡：显示与所选术语相关的术语

给术语添加分类(classification)

单击分类标签旁边的 + 可为术语添加分类。



# A checking account

Short Description: Short description

Long Description: Long description

Classifications: FINANCE\_PII  +

Categories: Accounts  +

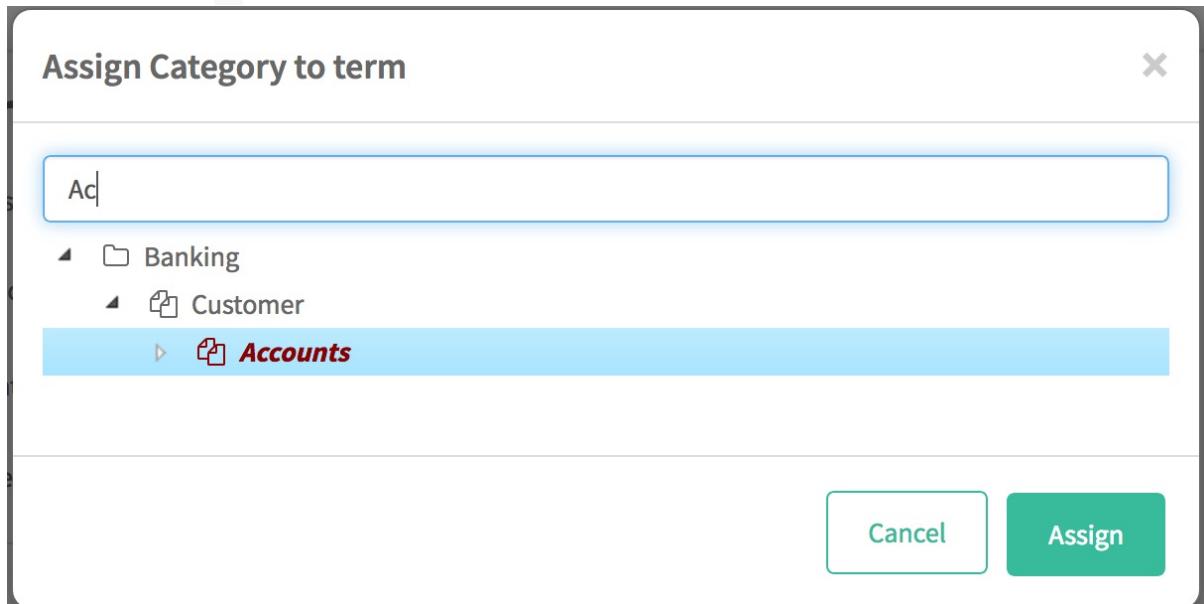
与其他术语建立术语关联

查看术语详细信息时，单击 Related Terms(相关术语) 选项卡。单击 + 将术语与当前术语链接。

Relation Types	Related Terms	Attributes
seeAlso		
synonyms		
antonyms		
preferredTerms		
preferredToTerms		

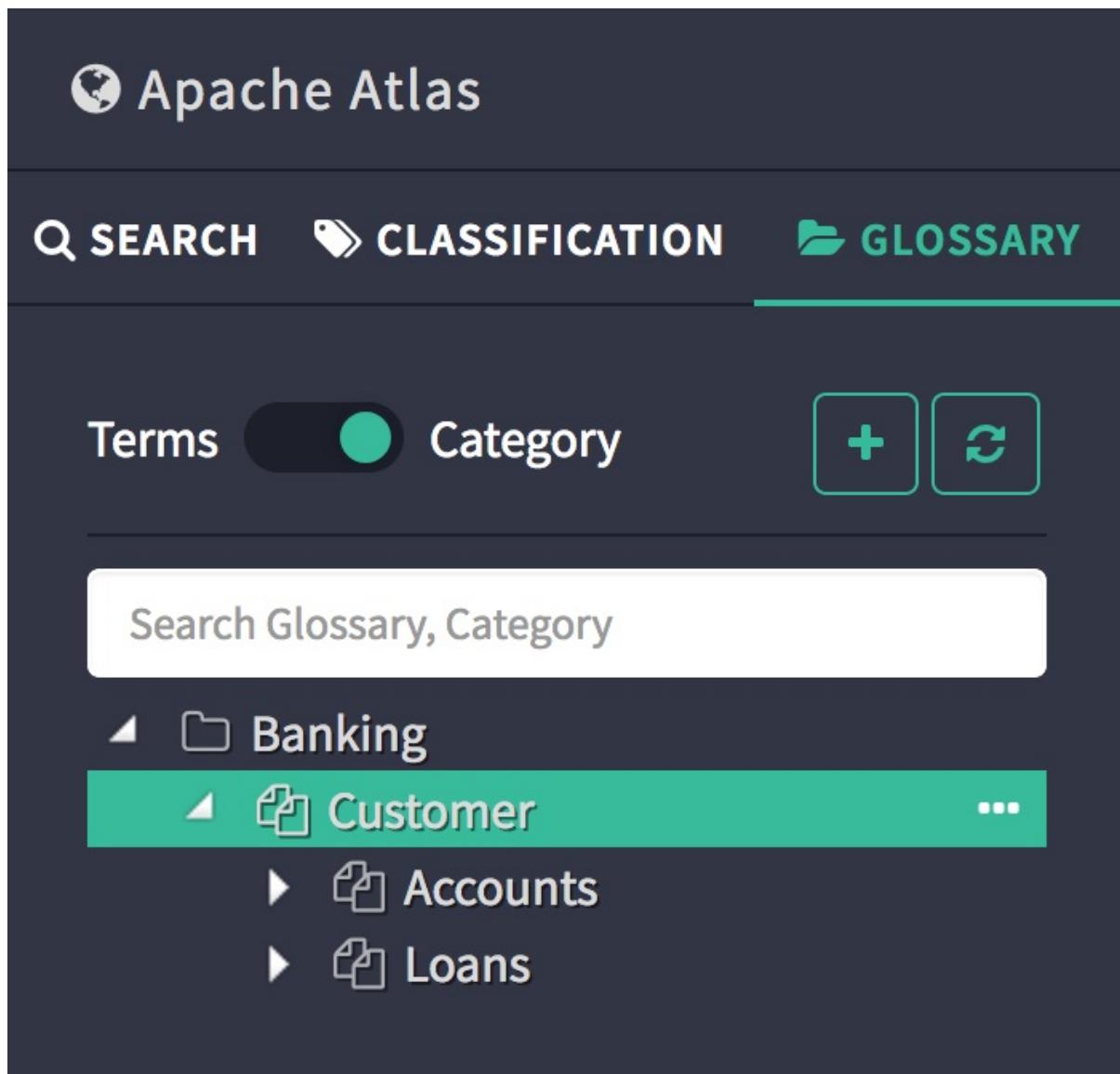
对术语进行分类

单击类别标签旁边的 + 可对术语进行分类。将提供模态对话框以选择类别。



## 类别视图(Category)

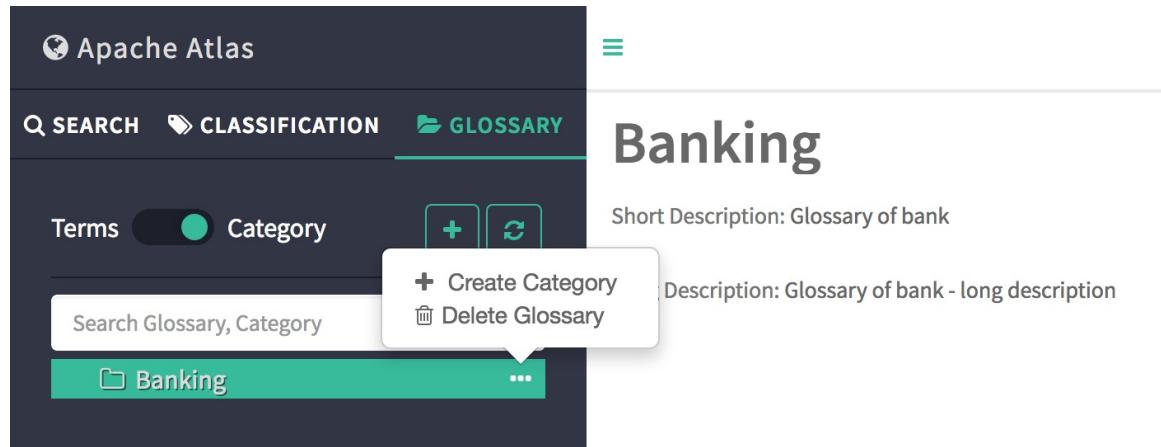
当切换开关处于 category 时，左侧面板将列出所有术语表以及类别层次结构。



类别菜单(Category)

单击 Category 旁边的省略号 ... 将显示类别上下文菜单。

- 创建新类别



- 创建子类别或删除类别

The screenshot shows the Apache Atlas interface. On the left, there's a sidebar with a tree view of categories: Banking, Customer, Accounts, and Loans. The 'Accounts' node is currently selected, highlighted in green. A context menu is open over the 'Accounts' node, containing options to 'Create Sub-Category' and 'Delete Category'. The right panel displays the details for the 'Loans' category, including its short description ('Loan categorization') and terms ('15-30 yr mortgage', 'ARM loans').

## 类别详情

选择 Category 后，详细信息将显示在右侧窗口中。

The screenshot shows the Apache Atlas interface. On the left, there's a sidebar with a tree view of categories: Banking, Customer, Accounts, and Loans. The 'Accounts' node is currently selected, highlighted in green. The right panel displays the details for the 'Accounts' category, including its short description ('Account categorization') and terms ('A savings account', 'A checking account').

## 术语分类

单击详情页中 Terms 标签旁边的 + 链接所选类别下的术语。

The screenshot shows the Apache Atlas interface. On the left, there's a sidebar with a tree view of categories: Banking, Customer, Accounts, and Loans. The 'Accounts' node is currently selected, highlighted in green. The right panel displays the details for the 'Accounts' category, including its short description ('Account categorization'), long description, and a 'Terms' section with a '+ Add Term' button.

**Assign term to Category**

Search Term

- Banking
  - 15-30 yr mortgage
  - A checking account
  - A savings account**
  - ARM loans

**Cancel** **Assign**

**Assign term to Category**

checkin

Banking
 

- A checking account**

**Cancel** **Assign**

## 术语分配流程

可以在搜索结果页和 `Glossary-Terms` 实体详情页中给 `entity(实体)` 分配术语。

### 分配术语

在搜索结果页面，点击 `terms` 列下的 +

The screenshot shows the Classification search interface. On the left, there are search filters for Type (hdfs\_path), Classification, Term, and Text. Below these are buttons for Clear, Save, and Search. On the right, a table displays 8 records from 1-25. The columns are Name, Owner, Description, Type, Classifications, and Term. Each row has a checkbox next to the Name. The 'Term' column contains a green '+' button. The table includes checkboxes for Exclude sub-types, Exclude sub-classifications, and Show historical entities, and a 'Columns' dropdown.

Name	Owner	Description	Type	Classifications	Term
/hive_data/hortonibank/ww_customers			hdfs_path	[+]	[+]
/hive_data/hortonibank/us_customers			hdfs_path	[+]	[+]
/hive_data/cost_savings			hdfs_path	[+]	[+]
/hive_data/finance/tax_2015			hdfs_path	[+]	[+]
/hive_data/hortonibank/eu_countries			hdfs_path	[+]	[+]
/hive_data/finance/tax_2009			hdfs_path	[+]	[+]
/hive_data/finance/tax_2010			hdfs_path	[+]	[+]
/hive_data/claim			hdfs_path	[+]	[+]

点击 terms 标签旁边的 +

The screenshot shows the term assignment interface for the entity '/hive\_data/hortonibank/ww\_customers'. It includes search filters for Type, Classification, Term, and Text, and buttons for Clear, Save, and Search. Below the search bar, it shows 'Classifications: [+]' and 'Term: [+ Add Term]'. The main area displays the entity path and a sidebar for assigning terms.

这两个操作都将显示下面的结果，按照屏幕上的提示完成术语分配。

The screenshot shows the 'Assign term to entity' dialog box. It has a search input field containing 'tax', a sidebar showing a tree structure with 'Banking' expanded and 'Tax' selected, and two buttons at the bottom: 'Cancel' and 'Assign'.

分类传播(Propagated classification)

如果一个术语具有分类，则该术语下的实体继承相同的分类。

**Claim**

Short Description: A request made by the insured for insurer remittance of payment due to loss incurred and covered under the policy agreement.

Long Description: A request made by the insured for insurer remittance of payment due to loss incurred and covered under the policy agreement.

Classifications: INSURANCE

Categories: hdfs\_path

Name	Owner	Description	Type	Classifications
/hive_data/cost_savings	hdfs_path			INSURANCE

## /hive\_data/cost\_savings (hdfs\_path)

Classifications: +

Term: Claim

Propagated Classifications: INSURANCE

Properties    Lineage    Relationships    **Classifications**    Audits

Showing 1 - 1

**Classification**

INSURANCE Propagated From

## 使用术语搜索

Apache Atlas基本搜索API和UI已更新，以支持术语作为搜索条件。允许用户查找与给定术语相关联的实体。

The screenshot shows the Apache Atlas search interface. At the top, there is a logo of a globe with the text "Apache Atlas". Below the logo are three navigation links: "SEARCH" (with a magnifying glass icon), "CLASSIFICATION" (with a tag icon), and "GLOSSARY" (with a book icon). A horizontal bar separates the header from the search controls. On the left, there are two buttons: "Basic" (selected, indicated by a green circle) and "Advanced" (indicated by a question mark icon). To the right of these buttons is a refresh icon in a rounded square. Below the search controls are four sections: "Search By Type" (dropdown menu "Select Type" with a filter icon), "Search By Classification" (dropdown menu "Select Classification" with a filter icon), "Search By Term" (dropdown menu "Search Term" with a filter icon), and "Search By Text" (text input field "Search by text"). At the bottom, there are two large buttons: "Clear" on the left and "Search" on the right.

Atlas支持以下操作，可[在这里](#)找到REST接口的详细信息。

### GlossaryREST

Show/Hide | List Operations | Expand Operations

<code>GET</code>	/v2/glossary/{glossaryGuid}/terms/headers	Get term headers belonging to a specific glossary.
<code>GET</code>	/v2/glossary/{glossaryGuid}/terms	Get terms belonging to a specific glossary.
<code>PUT</code>	/v2/glossary/{glossaryGuid}/partial	Partially update the glossary.
<code>GET</code>	/v2/glossary/{glossaryGuid}/detailed	Get a specific Glossary.
<code>GET</code>	/v2/glossary/{glossaryGuid}/categories/headers	Get the categories belonging to a specific glossary.
<code>GET</code>	/v2/glossary/{glossaryGuid}/categories	Get the categories belonging to a specific glossary.
<code>PUT</code>	/v2/glossary/{glossaryGuid}	Update the given glossary.
<code>GET</code>	/v2/glossary/{glossaryGuid}	Get a specific Glossary.
<code>DELETE</code>	/v2/glossary/{glossaryGuid}	Delete a glossary.
<code>GET</code>	/v2/glossary/terms/{termGuid}/related	Get all related terms for a specific term.
<code>POST</code>	/v2/glossary/terms/{termGuid}/assignedEntities	Assign the given term to the provided list of entity headers.
<code>GET</code>	/v2/glossary/terms/{termGuid}/assignedEntities	Get all entity headers assigned with the specified term.
<code>DELETE</code>	/v2/glossary/terms/{termGuid}/assignedEntities	Remove the term assignment for the given list of entity headers.
<code>POST</code>	/v2/glossary/terms	Create glossary terms in bulk.
<code>PUT</code>	/v2/glossary/term/{termGuid}/partial	Partially update the glossary term.
<code>PUT</code>	/v2/glossary/term/{termGuid}	Update the given glossary term.
<code>GET</code>	/v2/glossary/term/{termGuid}	Get specific glossary term.
<code>DELETE</code>	/v2/glossary/term/{termGuid}	Delete a glossary term.
<code>POST</code>	/v2/glossary/term	Create a glossary term.
<code>GET</code>	/v2/glossary/category/{categoryGuid}/terms	Get all terms associated with the specific category.
<code>GET</code>	/v2/glossary/category/{categoryGuid}/related	Get all related categories (parent and children).
<code>PUT</code>	/v2/glossary/category/{categoryGuid}/partial	Partially update the glossary category.
<code>PUT</code>	/v2/glossary/category/{categoryGuid}	Update the given glossary category.
<code>GET</code>	/v2/glossary/category/{categoryGuid}	Get specific glossary category.
<code>DELETE</code>	/v2/glossary/category/{categoryGuid}	Delete a glossary category.
<code>POST</code>	/v2/glossary/category	Create glossary category.
<code>POST</code>	/v2/glossary/categories	Create glossary category in bulk.
<code>POST</code>	/v2/glossary	Create a glossary.
<code>GET</code>	/v2/glossary	Retrieve all glossaries registered with Atlas.

### JSON结构

- Glossary

```
{
  "guid": "2f341934-f18c-48b3-aa12-eaa0a2bfce85",
  "qualifiedName": "SampleBank",
  "displayName": "Banking",
  "shortDescription": "Glossary of bank",
  "longDescription": "Glossary of bank - long description",
```

```

"language": "English",
"usage": "N/A",
"terms": [
{
    "termGuid": "502d34f1-b85f-4ad9-9d9f-fe7020ff0acb",
    "relationGuid": "6bb803e4-3af6-4924-aad6-6ad9f95ecd14",
    "displayText": "A savings account"
}, {
    "termGuid": "e441a540-ee55-4fc8-8eaf-4b9943d8929c",
    "relationGuid": "dbc46795-76ff-4f68-9043-be0eff0bc0f3",
    "displayText": "15-30 yr mortgage"
}, {
    "termGuid": "998e3692-51a8-47fe-b3a0-0d9f794437eb",
    "relationGuid": "0dc31b9-a81c-4185-ad4b-9209a97c305b",
    "displayText": "A checking account"
}, {
    "termGuid": "c4e2b956-2589-4648-8596-240d3bea5e44",
    "relationGuid": "e71c4a5d-694b-47a5-a41e-126ade857279",
    "displayText": "ARM loans"
}],
"categories": [
{
    "categoryGuid": "dd94859e-7453-4bc9-b634-a17fc14590f8",
    "parentCategoryGuid": "e6a3df1f-5670-4f9e-84da-91f77d008ce3",
    "relationGuid": "a0b7da02-1ccd-4415-bc54-3d0cdb8857e7",
    "displayText": "Accounts"
}, {
    "categoryGuid": "e6a3df1f-5670-4f9e-84da-91f77d008ce3",
    "relationGuid": "0e84a358-a4aa-4bd3-b806-497a6962ae1d",
    "displayText": "Customer"
}, {
    "categoryGuid": "7f041401-de8c-443f-a3b7-7bf5a910ff6f",
    "parentCategoryGuid": "e6a3df1f-5670-4f9e-84da-91f77d008ce3",
    "relationGuid": "7757b031-4e25-43a8-bf77-946f7f06c67a",
    "displayText": "Loans"
}]
}

```

## ● Term

```

{
    "guid": "e441a540-ee55-4fc8-8eaf-4b9943d8929c",
    "qualifiedName": "fixed_mtg@SampleBank",
    "displayName": "15-30 yr mortgage",
    "shortDescription": "Short description",
    "longDescription": "Long description",
    "examples": ["N/A"],
    "abbreviation": "FMTG",
    "anchor": {
        "glossaryGuid": "2f341934-f18c-48b3-aa12-eaa0a2bfce85",
        "relationGuid": "dbc46795-76ff-4f68-9043-be0eff0bc0f3"
    },
    "categories": [
{
    "categoryGuid": "7f041401-de8c-443f-a3b7-7bf5a910ff6f",
    "relationGuid": "b4cd33-7b0c-41e2-9324-afe549ec6ada",
    "displayText": "Loans"
}],
    "seeAlso": [],
    "synonyms": [],
    "antonyms": [],
    "replacedBy": [],
    "replacementTerms": [],
    "translationTerms": [],
    "translatedTerms": [],
    "isA": [],
    "classifies": [],
    "preferredTerms": [],
    "preferredToTerms": [
{

```

```

        "termGuid" : "c4e2b956-2589-4648-8596-240d3bea5e44",
        "displayText": "ARM Loans"
    }]
}

```

- Category

```

{
  "guid": "7f041401-de8c-443f-a3b7-7bf5a910ff6f",
  "qualifiedName": "Loans.Customer@HortonBank",
  "displayName": "Loans",
  "shortDescription": "Loan categorization",
  "anchor": {
    "glossaryGuid": "2f341934-f18c-48b3-aa12-eaa0a2bfce85",
    "relationGuid": "7757b031-4e25-43a8-bf77-946f7f06c67a"
  },
  "parentCategory": {
    "categoryGuid": "e6a3df1f-5670-4f9e-84da-91f77d008ce3",
    "relationGuid": "8a0a8e11-0bb5-483b-b7d6-cfe0b1d55ef6"
  },
  "childrenCategories" : [],
  "terms": [
    {
      "termGuid": "e441a540-ee55-4fc8-8eaf-4b9943d8929c",
      "relationGuid": "b4cddd33-7b0c-41e2-9324-afe549ec6ada",
      "displayText": "15-30 yr mortgage"
    },
    {
      "termGuid": "c4e2b956-2589-4648-8596-240d3bea5e44",
      "relationGuid": "8db1e784-4f04-4eda-9a58-6c9535a95451",
      "displayText": "ARM loans"
    }
  ]
}

```

## 创建操作(CREATE)

1. 创建术语表
2. 创建一个术语
3. 创建分类术语
4. 用关系创建术语
5. 创建一个类别
6. 创建具有层次结构的类别
7. 创建类别并对术语进行分类
8. 为实体分配术语

注意:

- 在创建操作期间，术语表、术语和类别将获得自动分配的GUID和qualifiedName。
- 要创建包含子项的类别，必须事先创建子项。
- 要创建属于某个类别的术语，必须事先创建该类别。
- 要创建关系术语，必须事先创建相关术语。

## 读操作(READ)

1. 通过GUID获取术语表 - 提供属于术语表的所有术语和类别（标题）。
2. 获取所有术语表 - 为所有术语表提供他们的术语和类别（标题）。
3. 通过GUID获取术语 - 提供有关术语，其所属类别（如果有）以及任何相关术语的详细信息。
4. 通过GUID获取类别 - 提供有关类别，类别层次结构（如果有）和属于该类别的术语的详细信息。
5. 获取给定术语表的所有术语 - 提供属于给定术语表的所有术语（具有 # 3中提到的详细信息）。
6. 获取给定术语表的所有类别 - 提供属于给定术语表的所有类别（具有 # 4中提到的详细信息）。
7. 获取与给定术语相关的所有术语 - 提供与给定术语相关/链接的所有术语。
8. 获取与给定类别（父母和子女）相关的所有类别

## 9. 获取给定类别的所有条款

### 更新操作(UPDATE)

1. 局部更新术语表
2. 局部更新术语
3. 局部更新类别
4. 更新给定的词汇表
5. 更新给定的术语
6. 更新给定的类别

注意：

- 局部更新仅处理词汇表模型文件中定义的原始属性。
- 分配后，无法更改GUID和qualifiedName。唯一的方法是删除并重新创建所需的对象。
- 在任何更新中都无法删除锚点
- 更新API期望在GET调用之后就地修改JSON。任何缺失的属性/关系都将被删除。
- 对类别层次结构的任何更新都会导致对其下的层次结构进行级联更新，例如锚更改会影响所有子项，父项更改会影响self和children的qualifiedName。

### 删除操作(DELETE)

1. 删除术语表 - 删除锚定到给定词汇表的所有类别和术语。如果已为实体分配任何术语，则会阻止此删除。
2. 删除术语 - 仅当术语未与任何实体关联/分配时才删除该术语。
3. 删除类别 - 仅删除给定类别，所有子项都成为顶级类别。
4. 从实体中删除术语分配

- 06. 安全
  - 概述
    - SSL
    - 服务认证
    - JAAS 配置
    - 基于SPNEGO的HTTP身份验证

## 06. 安全

### 概述

以下功能可用于增强平台的安全性：

- SSL
- 服务认证
- 基于SPNEGO的HTTP身份验证

### SSL

支持SSL单向（服务器身份验证）和双向（服务器和客户端身份验证）。以下应用程序属性（atlas-application.properties文件中配置的属性）可用于配置SSL：

- `atlas.enableTLS` (false | true) [default: false]: 启用/禁用SSL侦听器。
- `keystore.file`：服务器利用的密钥库文件的路径。该文件包含服务器证书。
- `truststore.file`：信任库文件的路径。此文件包含其他可信实体的证书（例如，如果启用了双向SSL，则为客户端进程的证书）。在大多数情况下，可以将其设置为与`keystore.file`属性相同的值（特别是如果启用了单向SSL）。
- `client.auth.enabled` (false | true) [default: false]: 启用/禁用客户端身份验证。如果启用，客户端将必须在传输会话密钥创建过程期间向服务器进行身份验证（即双向SSL有效）。
- `cert.stores.credential.provider.path`：凭据提供程序存储文件的路径。密钥库，信任库和服务器证书的密码在此安全文件中维护。利用'bin'nirectoy中的cputil脚本（见下文），用所需的密码填充此文件。
- `atlas.ssl.exclude.cipher.suites`：排除的密码套件列表 - NULL.,,RC4.,,MD5.,,DES.,,DSS.是弱且不安全的密码套件默认排除。如果需要排除其他密码，请使用默认的密码套件设置此属性，例如`atlas.ssl.exclude.cipher.suites =.NULL.,,RC4.,,MD5.,,DES.,,DSS.` DSS。\*，并使用逗号分隔符将其他Ciper Suites添加到列表中。可以使用其全名或正则表达式添加它们。 --- `atlas.ssl.exclude.cipher.suites`属性中列出的密码套件将优先于默认的密码套件。一个人会保留默认的密码套件，并添加额外的密码套件以确保安全。

### 凭据提供实用脚本

为了防止使用明文密码，Atlas使用Credential Provider工具进行安全密码存储（有关此工具的更多信息，请参阅[Hadoop凭据命令参考](#)）。可以利用'bin'目录中的cputil脚本来创建所需的密码存储。

要为Atlas创建凭据提供程序：

- cd到'bin'目录
- 输入 `./cputil.py`
- 输入生成的凭据提供程序的路径。路径的格式为：
  - `jceks: //file/local/file/path/file.jceks` 或 `jceks://namenodehost: port/path/in/hdfs/to/ file.jceks`。这些文件通常使用".jceks"扩展名（例如test.jceks）
- 输入密钥库、信任库和服务器密钥的密码（这些密码需要与用于实际创建关联证书存储文件的密码相匹配）。

将生成凭证提供程序并将其保存到提供的路径中。

## 服务认证

Atlas平台在启动时与经过身份验证的身份相关联。默认情况下，在不安全的环境中，该标识与启动服务器的OS身份验证用户相同。但是，在利用kerberos的安全集群中，最佳做法是配置密钥表和主体，以便平台对KDC进行身份验证。这允许服务随后与其他安全集群服务（例如HDFS）交互。

配置服务身份验证的属性包括：

- `atlas.authentication.method` (`simple | kerberos`) [default: `simple`]: 要使用的身份验证方法。Simple将利用OS身份验证身份并且是默认机制。`'kerberos'`表示该服务需要使用已配置的密钥表和主体对KDC进行身份验证。
- `atlas.authentication.keytab` : keytab文件的路径。
- `atlas.authentication.principal` : 用于向KDC进行身份验证的主体。主体通常是“`user / host @ realm`”形式。您可以使用`_HOST`标记作为主机名，本地主机名将由运行时替换（例如“`Atlas/_HOST@EXAMPLE.COM`”）。请注意，当Atlas配置了HBase作为安全集群中的存储后端时，图形db（JanusGraph）需要足够的用户权限才能创建和访问HBase表。要授予适当的权限，请参阅[图形持久性引擎-Hbase](#)。

## JAAS 配置

在安全集群中，Atlas与之交互的一些组件（例如Kafka）需要Atlas使用JAAS向他们进行身份验证。以下属性用于设置适当的JAAS配置。

- `atlas.jaas.client-id.loginModuleName` - the authentication method used by the component (for example, `com.sun.security.auth.module.Krb5LoginModule`)
- `atlas.jaas.client-id.loginModuleControlFlag` - (required|requisite|sufficient|optional) [default: required]
- `atlas.jaas.client-id.option.useKeyTab` (true|false)
- `atlas.jaas.client-id.option.storeKey` (true | false)
- `atlas.jaas.client-id.option.serviceName` - service name of server component
- `atlas.jaas.client-id.option.keyTab` =
- `atlas.jaas.client-id.option.principal` =

例如，jaas-application.properties文件中的以下属性设置：

```
atlas.jaas.KafkaClient.loginModuleName = com.sun.security.auth.module.Krb5LoginModule
atlas.jaas.KafkaClient.loginModuleControlFlag = required
atlas.jaas.KafkaClient.option.useKeyTab = true
atlas.jaas.KafkaClient.option.storeKey = true
atlas.jaas.KafkaClient.option.serviceName = kafka
atlas.jaas.KafkaClient.option.keyTab = /etc/security/keytabs/kafka_client.keytab
atlas.jaas.KafkaClient.option.principal = kafka-client-1@EXAMPLE.COM

atlas.jaas.MyClient.0.loginModuleName = com.sun.security.auth.module.Krb5LoginModule
atlas.jaas.MyClient.0.loginModuleControlFlag = required
atlas.jaas.MyClient.0.option.useKeyTab = true
atlas.jaas.MyClient.0.option.storeKey = true
atlas.jaas.MyClient.0.option.serviceName = kafka
atlas.jaas.MyClient.0.option.keyTab = /etc/security/keytabs/kafka_client.keytab
atlas.jaas.MyClient.0.option.principal = kafka-client-1@EXAMPLE.COM

atlas.jaas.MyClient.1.loginModuleName = com.sun.security.auth.module.Krb5LoginModule
atlas.jaas.MyClient.1.loginModuleControlFlag = optional
atlas.jaas.MyClient.1.option.useKeyTab = true
atlas.jaas.MyClient.1.option.storeKey = true
atlas.jaas.MyClient.1.option.serviceName = kafka
atlas.jaas.MyClient.1.option.keyTab = /etc/security/keytabs/kafka_client.keytab
atlas.jaas.MyClient.1.option.principal = kafka-client-1@EXAMPLE.COM
```

该配置等同于以下jaas.conf文件条目：

```
KafkaClient {
```

```

com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
storeKey=true
serviceName=kafka
keyTab="/etc/security/keytabs/kafka_client.keytab"
principal="kafka-client-1@EXAMPLE.COM";
};

MyClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    serviceName=kafka keyTab="/etc/security/keytabs/kafka_client.keytab"
    principal="kafka-client-1@EXAMPLE.COM";
};

MyClient {
    com.sun.security.auth.module.Krb5LoginModule optional
    useKeyTab=true
    storeKey=true
    serviceName=kafka
    keyTab="/etc/security/keytabs/kafka_client.keytab"
    principal="kafka-client-1@EXAMPLE.COM";
};

```

## 基于SPNEGO的HTTP身份验证

通过启用平台的SPNEGO支持，可以保护对Atlas平台的HTTP访问。目前有两种支持的身份验证机制：

- `simple`：通过提供的用户名执行身份验证。
- `Kerberos`：利用客户端的KDC身份验证身份对服务器进行身份验证。

Kerberos支持要求访问服务器的客户端首先向KDC进行身份验证（通常这是通过'kinit'命令完成的）。一旦经过身份验证，用户就可以访问服务器（身份验证是通过SPNEGO协商机制与服务器交互）。

配置SPNEGO支持的属性包括：

- `atlas.http.authentication.enabled` (`true` | `false`) [default: `false`]：是否启用HTTP身份验证的属性
- `atlas.http.authentication.type` (`simple` | `kerberos`) [default: `simple`]：身份验证类型
- `atlas.http.authentication.kerberos.principal`：Web应用程序Kerberos主体名称。Kerberos主体名称必须以“HTTP / ...”开头。例如：“HTTP / localhost @ LOCALHOST”。没有默认值。
- `atlas.http.authentication.kerberos.keytab` - 包含kerberos主体凭据的keytab文件的路径。
- `atlas.rest.address` :: // :

有关HTTP身份验证机制的更详细讨论，请参阅 [Hadoop Auth, Java HTTP SPNEGO 2.6.0 - 服务器端配置](#)。在Atlas身份验证实现的情况下，文档引用的前缀是 `atlas.http.authentication`。

## 客户端安全配置

当Atlas客户端通过代码，使用SSL传输和/或Kerberos身份验证方式与Atlas服务端通信时，需要提供Atlas客户端配置文件，该文件提供允许与服务器通信或进行身份验证的安全属性。使用适当的设置更新`atlas-application.properties`文件（参见下文），并将其复制到客户端的类路径或“`atlas.conf`”系统属性指定的目录。

SSL通信相关的客户端属性有：

- `atlas.enableTLS` (`false` | `true`) [default: `false`]：启用/禁用SSL客户端通信基础结构。
- `keystore.file`：客户端利用的密钥库文件的路径。仅当在服务器上启用了双向SSL并且包含客户端证书时，才需要此文件。
- `truststore.file`：信任库文件的路径。此文件包含可信实体的证书（例如，服务器或共享证书颁发机构的证书）。单向或双向SSL都需要此文件。
- `cert.stores.credential.provider.path`：凭据提供程序存储文件的路径。密钥库，信任库和客户端证书的密码在此安全文件中维护。验证服务器所需的属性（如果启用了身份验证）：

- atlas.http.authentication.type (simple | kerberos) [default: simple]: 身份验证类型

## SOLR Kerberos 配置

如果指定的身份验证类型为“kerberos”，则将访问kerberos票证缓存以向服务器进行身份验证（因此，客户端需要在使用“kinit”或类似机制与服务器通信之前向KDC进行身份验证）。

参考 [the Apache SOLR Kerberos configuration](#)

- 添加主体并生成solr的keytab文件。为每个要运行Solr的主机为每个主机创建一个keytab文件，并将主体名称与主机一起使用（例如：addprinc -randkey solr/\${HOST1}@EXAMPLE.COM。将\${HOST1}替换为实际的主机名）。

```
kadmin.local
kadmin.local: addprinc -randkey solr/<hostname>@EXAMPLE.COM
kadmin.local: xst -k solr.keytab solr/<hostname>@EXAMPLE.COM
kadmin.local: quit
```

- 添加主体并生成用于验证HTTP请求的keytab文件。（请注意，如果Ambari用于Kerberize集群，则可以使用keytab /etc/security/keytabs/spnego.service.keytab）

```
kadmin.local
kadmin.local: addprinc -randkey HTTP/<hostname>@EXAMPLE.COM
kadmin.local: xst -k HTTP.keytab HTTP/<hostname>@EXAMPLE.COM
kadmin.local: quit
```

- 将keytab文件复制到运行Solr的所有主机。

```
cp solr.keytab /etc/security/keytabs/
chmod 400 /etc/security/keytabs/solr.keytab

cp HTTP.keytab /etc/security/keytabs/
chmod 400 /etc/security/keytabs/HTTP.keytab
```

- 在Zookeeper中创建路径以存储Solr配置和其他参数。

```
$SOLR_INSTALL_HOME/server/scripts/cloud-scripts/zkcli.sh -zkhost $ZK_HOST:2181 -cmd makepath solr
```

- 将配置上传到Zookeeper

```
$SOLR_INSTALL_HOME/server/scripts/cloud-scripts/zkcli.sh -cmd upconfig -zkhost $ZK_HOST:2181/solr -confname basic_configs -confdir $SOLR_INSTALL_HOME/server/solr/configsets/basic_configs/conf
```

- 创建JAAS配置

```
vi /etc/solr/conf/solr_jaas.conf

Client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="/etc/security/keytabs/solr.keytab"
    storeKey=true
    useTicketCache=true
    debug=true
    principal="solr/<hostname>@EXAMPLE.COM";
};
```

- 将 /etc/solr/conf/solr\_jaas.conf 复制到运行Solr的所有主机。

- 在 \$SOLR\_INSTALL\_HOME/bin/ 中编辑 solr.in.sh

```
vi $SOLR_INSTALL_HOME/bin/solr.in.sh

SOLR_JAAS_FILE=/etc/solr/conf/solr_jaas.conf
SOLR_HOST=`hostname -f`
ZK_HOST="$ZK_HOST1:2181,$ZK_HOST2:2181,$ZK_HOST3:2181/solr"
KERBEROS_REALM="EXAMPLE.COM"
SOLR_KEYTAB=/etc/solr/conf/solr.keytab
SOLR_KERB_PRINCIPAL=HTTP@${KERBEROS_REALM}
SOLR_KERB_KEYTAB=/etc/solr/conf/HTTP.keytab
SOLR_AUTHENTICATION_CLIENT_CONFIGURER="org.apache.solr.client.solrj.impl.Krb5HttpClientConfigurer"
SOLR_AUTHENTICATION_OPTS="-DauthenticationPlugin=org.apache.solr.security.KerberosPlugin -Djava.security.auth.login.config=${SOLR_JAAS_FILE} -Dsolr.kerberos.principal=${SOLR_KERB_PRINCIPAL} -Dsolr.kerberos.keytab=${SOLR_KERB_KEYTAB} -Dsolr.kerberos.cookie.domain=${SOLR_HOST} -Dhost=${SOLR_HOST} -Dsolr.kerberos.name.rules=DEFAULT"
```

- 将solr.in.sh复制到运行Solr的所有主机。
- 设置Solr以通过上传 security.json 来使用Kerberos插件。

```
$SOLR_INSTALL_HOME/server/scripts/cloud-scripts/zkcli.sh -zkhost <zk host>:2181 -cmd put /security.json '{"authentication":{"class": "org.apache.solr.security.KerberosPlugin"}}'
```

- 启动Solr: `` \$SOLR\_INSTALL\_HOME/bin/solr start -cloud -z \$ZK\_HOST1:2181,\$ZK\_HOST2:2181,\$ZK\_HOST3:2181 -noprompt

- 测试Solr:

```
kinit -k -t /etc/security/keytabs/HTTP.keytab HTTP/<host>@EXAMPLE.COM curl --negotiate -u : "http://:8983/solr/" ````
```

- 在Solr中创建与Atlas使用的索引相对应的集合，并将Atlas配置更改为指向Solr实例设置，参考此处 [安装步骤](#)。

- 07. 认证
  - Authentication
    - FILE认证方式
    - Kerberos 认证方式
    - LDAP 认证方式

## 07. 认证

### Authentication

Atlas 支持以下的认证方式：

- File
- Kerberos
- LDAP

在 `atlas-application.properties` 文件中启用该类型的身份验证，以下属性设置为true：

```
atlas.authentication.method.kerberos=true|false
atlas.authentication.method.ldap=true|false
atlas.authentication.method.file=true|false
```

如果两个或多个身份验证方法设置为true，则如果之前的身份验证失败，则身份验证将回退到后一种方法。例如，如果 Kerberos 身份验证设置为true并且ldap身份验证也设置为true，那么，如果对于没有kerberos principal和keytab的请求，LDAP身份验证将用作回退方案。

#### FILE认证方式

文件身份验证要求用户凭据文件中的用户登录详细信息采用下面指定的格式，文件路径应设置为 `atlas-application.properties` 中的属性 `atlas.authentication.method.file.filename`。

```
atlas.authentication.method.file=true
atlas.authentication.method.file.filename=${sys:atlas.home}/conf/users-credentials.properties
```

用户凭证文件应具有以下格式：

```
username=group::sha256-password
```

例如：

```
admin=ADMIN::e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221a
```

用户组可以是ADMIN, DATA\_STEWARD或DATA\_SCIENTIST 注意：`-password` 使用sha256编码方法编码，可以使用unix工具生成。

例如：

```
echo -n "Password" | sha256sum
e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221a -
```

#### Kerberos 认证方式

要在Atlas中以Kerberos模式启用身份验证，请在 `atlas-application.properties` 中将属性 `atlas.authentication.method.kerberos` 设置为true：

```
atlas.authentication.method.kerberos = true
```

还需要设置以下属性：

```
atlas.authentication.method.kerberos.principal=<principal>/<fqdn>@EXAMPLE.COM
atlas.authentication.method.kerberos.keytab = /<key tab filepath>.keytab
atlas.authentication.method.kerberos.name.rules = RULE:[2:$1@$0](atlas@EXAMPLE.COM)s/.*/atlas/
atlas.authentication.method.kerberos.token.validity = 3600 [ in Seconds (optional)]
```

## LDAP 认证方式

要在Atlas中以LDAP模式启用身份验证，请将属性 `atlas.authentication.method.ldap` 设置为true，并将属性 `atlas.authentication.method.ldap.type` 的Ldap类型设置为 `atlas-application.properties` 中的LDAP或AD。如果连接到Active Directory，请使用AD。

```
atlas.authentication.method.ldap=true
atlas.authentication.method.ldap.type=ldap|ad
```

对于LDAP或AD，需要在atlas应用程序属性中设置以下配置。

### Active Directory

```
atlas.authentication.method.ldap.ad.domain= example.com
atlas.authentication.method.ldap.ad.url=ldap://<AD server ip>:389
atlas.authentication.method.ldap.ad.base_dn=DC=example,DC=com
atlas.authentication.method.ldap.ad.bind_dn=CN=Administrator,CN=Users,DC=example,DC=com
atlas.authentication.method.ldap.ad.bind.password=<password>
atlas.authentication.method.ldap.ad.referral=ignore
atlas.authentication.method.ldap.ad.user.searchfilter=(sAMAccountName={0})
atlas.authentication.method.ldap.ad.default.role=ROLE_USER
```

### LDAP Directroy

```
atlas.authentication.method.ldap.url=ldap://<Ldap server ip>:389
atlas.authentication.method.ldap.userDNpattern=uid={0},ou=users,dc=example,dc=com
atlas.authentication.method.ldap.groupSearchBase=dc=example,dc=com
atlas.authentication.method.ldap.groupSearchFilter=(member=cn={0},ou=users,dc=example,dc=com)
atlas.authentication.method.ldap.groupRoleAttribute=cn
atlas.authentication.method.ldap.base_dn=dc=example,dc=com
atlas.authentication.method.ldap.bind_dn=cn=Manager,dc=example,dc=com
atlas.authentication.method.ldap.bind.password=<password>
atlas.authentication.method.ldap.referral=ignore
atlas.authentication.method.ldap.user.searchfilter=(uid={0})
atlas.authentication.method.ldap.default.role=ROLE_USER
```

- 08. 授权模型

## 08. 授权模型

### 介绍

Atlas是一组可伸缩且可扩展的核心基础治理服务 - 使企业能够在Hadoop内有效地满足其合规性要求，并允许与整个企业数据生态系统集成。Apache Atlas为组织提供开放式元数据管理和治理功能，以构建其数据资产目录，对这些资产进行分类和管理，并为数据科学家，分析师和数据治理团队提供围绕这些数据资产的协作功能。

本文档介绍了Apache Atlas支持的授权模型的详细信息，以控制对Atlas管理的元数据的访问。

### 授权访问类型

Apache Atlas提供了一种类型(Type)系统，允许用户对他们想要管理的元数据对象进行建模。该模型由称为“类型”的定义组成。Apache Atlas类型系统支持以下种类的类型：

- Entity
- Classification
- Relationship
- Struct
- Enum

授权模型允许控制哪些用户，组可以根据类型名称和类型类别对类型执行以下操作：

- create
- update
- delete

以下是模型支持的访问控制的几个示例：

- 管理员用户可以create/update/delete所有类别的类型
- 数据管理员可以create/update/delete分类类型
- 健康检查(Healthcare)数据管理员可以create/update/delete名称以“hc”开头的类型

### 授权访问实体

实体是Entity类型的实例，此类实例表示现实世界中的对象：例如Hive中的表，HDFS文件，Kafka主题。授权模型可以控制哪些用户，组可以对实体执行以下操作（基于实体类型，实体分类，实体ID）：

- read
- create
- update
- delete
- read classification
- add classification
- update classification
- remove classification

以下是模型支持的访问控制的几个示例：

- 管理员用户可以对所有类型的实体执行所有实体操作
- 数据管理员可以对所有类型的实体执行除删除之外的所有实体操作
- 数据质量管理员可以add/update/remove DATA\_QUALITY分类
- 特定组中的用户可以使用PII分类或其子分类来read/update实体

- 财务用户可以read/update ID以“finance”开头的实体

## 管理员操作的授权

授权模型可以控制哪些用户、组可以执行以下管理操作：

- 导入实体
- 导出实体

具有上述访问权限的用户可以导入/导出实体，而无需授予它们细粒度的实体级访问权限。

## 可插拔授权

Apache Atlas支持可插入的授权接口，如下所示，启用备用实现来处理授权。

可以使用配置 `atlas.authorizer.impl` 向Apache Atlas注册实现授权接口的类的名称。如果未设置此属性，Apache Atlas将在 `org.apache.atlas.authorize.simple.AtlasSimpleAuthorizer` 中使用其默认实现。

```
package org.apache.atlas.authorize;

public interface AtlasAuthorizer {
    void init();

    void cleanUp();

    boolean isAccessAllowed(AtlasAdminAccessRequest request) throws AtlasAuthorizationException;

    boolean isAccessAllowed(AtlasEntityAccessRequest request) throws AtlasAuthorizationException;

    boolean isAccessAllowed(AtlasTypeAccessRequest request) throws AtlasAuthorizationException;
}
```

## 简单授权

简单授权程序是Apache Atlas中包含的默认授权程序实现。有关设置Apache Atlas以使用简单授权程序的详细信息，请参阅后面章节：设置Atlas以使用Simple Authorizer。

## Ranger授权

要将Apache Atlas配置为使用Apache Ranger提供的授权实现，请在`application.properties`配置文件中包含以下属性：

```
atlas.authorizer.impl=ranger
```

Apache Ranger Authorizer需要设置配置文件，例如指定Apache Ranger管理服务器URL，包含授权策略的服务名称等。有关详细信息，请参阅后面章节：设置Atlas以使用Ranger Authorizer。

## 无授权

除了默认授权程序之外，Apache Atlas还包括一个允许所有用户访问的授权程序。此授权程序在测试环境和单元测试中非常有用。要使用此授权程序，请设置以下配置：

```
atlas.authorizer.impl=NONE
```



- 8.1 Atlas简单授权

## 8.1 Atlas简单授权

如Atlas Authorization Model中所详述的，Apache Atlas支持可插拔授权模型。简单授权程序是Apache Atlas中包含的默认授权程序实现。简单授权程序使用JSON文件中定义的策略。本文档提供了配置Apache Atlas以使用简单授权程序的步骤的详细信息，以及包含授权策略的JSON文件格式的详细信息。

### 配置

要将Apache Atlas配置为使用简单授权程序，请在application.properties配置文件中包含以下属性：

```
atlas.authorizer.impl=simple
atlas.authorizer.simple.authz.policy.file=/etc/atlas/conf/atlas-simple-authz-policy.json
```

请注意，如果指定的策略文件位置不是绝对路径，则将在以下路径中查找该文件：

- Apache Atlas配置目录（由系统属性atlas.conf指定）
- Apache Atlas服务器的当前目录
- CLASSPATH

### 策略文件格式

简单授权程序使用角色(roles)对权限进行分组，然后可以将权限分配给用户和用户组。以下示例有助于理解策略文件格式的详细信息：

### 角色

以下策略文件定义了3个角色：

- ROLE\_ADMIN：拥有所有权限
- PROD\_READ\_ONLY：可以访问具有以“@prod”结尾的qualifiedName的读取实体
- TEST\_ALL\_ACCESS：具有对以“@test”结尾的qualifiedName的实体的所有访问权限

简单授权程序支持Java reg-ex指定privilege / entity-type / entity-id / classification / typeName / typeCategory的值。

```
{
  "roles": {
    "ROLE_ADMIN": {
      "adminPermissions": [
        {
          "privileges": [ ".*" ]
        }
      ],
      "entityPermissions": [
        {
          "privileges": [ ".*" ],
          "entityTypes": [ ".*" ],
          "entityIds": [ ".*" ],
          "classifications": [ ".*" ]
        }
      ],
      "typePermissions": [
        {
          "privileges": [ ".*" ],
          "typeCategories": [ ".*" ],
          "typeNames": [ ".*" ]
        }
      ]
    }
  }
}
```

```

        "typeNames": [ ".*" ]
    },
],
},
"PROD_READ_ONLY" : {
    "entityPermissions": [
        {
            "privileges": [ "entity-read", "entity-read-classification" ],
            "entityTypes": [ ".*" ],
            "entityIds": [ ".*@prod" ],
            "classifications": [ ".*" ]
        }
    ]
},
"TEST_ALL_ACCESS" : {
    "entityPermissions": [
        {
            "privileges": [ ".*" ],
            "entityTypes": [ ".*" ],
            "entityIds": [ ".*@test" ],
            "classifications": [ ".*" ]
        }
    ]
},
"userRoles": {
    ...
},
"groupRoles": {
    ...
}
}

```

## 将角色分配给用户和用户组

上面定义的角色可以分配（授予）给用户，如下所示：

```

{
    "roles": {
        ...
    },
    "userRoles": {
        "admin": [ "ROLE_ADMIN" ],
        "steward": [ "DATA_STEWARD" ],
        "user1": [ "PROD_READ_ONLY" ],
        "user2": [ "TEST_ALL_ACCESS" ],
        "user3": [ "PROD_READ_ONLY", "TEST_ALL_ACCESS" ],
    },
    "groupRoles": {
        ...
    }
}

```

可以将角色分配（授予）给用户组，如下所示。用户可以属于多个组;分配给用户所属的所有组的角色将用于授权访问。

```
{
    "roles": {
        ...
    },
}
```

```
"userRoles": {  
    ...  
},  
  
"groupRoles": {  
    "admins": [ "ROLE_ADMIN" ],  
    "dataStewards": [ "DATA_STEWARD" ],  
    "testUsers": [ "TEST_ALL_ACCESS" ],  
    "prodReadUsers": [ "PROD_READ_ONLY" ]  
}  
}
```

- 8.2 Atlas Ranger授权

## 8.2 Atlas Ranger授权

如Atlas Authorization Model中所详述的，Apache Atlas支持可插拔授权模型。Apache Ranger提供了一个使用Apache Ranger策略进行授权的授权器实现。此外，Apache Ranger提供的授权程序会到中央审计仓库进行审计(Audit)。

### 配置

要配置Apache Atlas以使用Apache Ranger授权程序，请按照以下说明操作：在 `atlas-application.properties` 配置文件中包含以下属性：

```
atlas.authorizer.impl=ranger
```

如果您使用Apache Ambari部署Apache Atlas和Apache Ranger，请在Apache Ranger的配置页面中启用Atlas插件。

- 在Apache Atlas的libext目录中包含Apache Ranger插件库
  - /libext/ranger-atlas-plugin-impl/
  - /libext/ranger-atlas-plugin-shim-.jar
  - /libext/ranger-plugin-classloader-.jar
- 在Apache Atlas的配置目录中包含Apache Ranger插件的配置文件 - 通常位于 `/etc/atlas/conf` 目录下。有关配置文件内容的更多详细信息，请参阅Apache Ranger中的相应文档。
  - /ranger-atlas-audit.xml
  - /ranger-atlas-security.xml
  - /ranger-policymgr-ssl.xml
  - /ranger-security.xml

### Apache Ranger授权策略模型

Apache Atlas的Apache Ranger授权策略模型支持3个资源层次结构，以控制对类型，实体和管理操作的访问。以下图像显示了Apache Ranger中每种策略的各种详细信息。

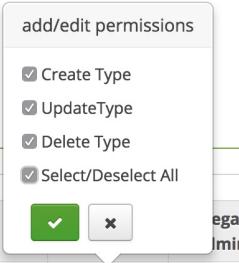
- 类型 遵循授权策略允许用户'admin'创建/更新/删除任何分类类型。

**Policy Details :**

Policy Type **Access** 

Policy Name *	<input type="text" value="All Classifications"/>	<input checked="" type="radio"/> enabled <input type="radio"/> normal
Policy Label	<input type="text" value="Policy Label"/>	
type-category *	<input type="text" value="classification"/>	<input checked="" type="radio"/> include
Type Name *	<input type="text" value="*"/>	<input checked="" type="radio"/> include
Description	<input type="text" value="Authorization to create, update, delete classifications"/>	
Audit Logging	<input checked="" type="radio"/> YES	

**Allow Conditions :**

Select Group	Select User	Add Permissions
<input type="text" value="Select Group"/>	<input type="text" value="admin"/>	

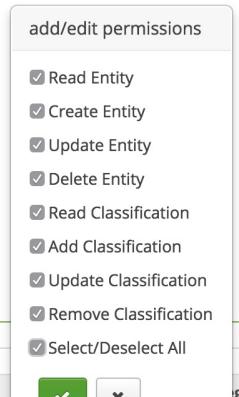
- 实体 遵循授权策略允许用户'admin'对名为“my\_db”的Hive数据库的元数据实体执行所有操作。

**Policy Details :**

Policy Type **Access** 

Policy Name *	<input type="text" value="Hive entities for my_db"/>	<input checked="" type="radio"/> enabled <input type="radio"/> normal
Policy Label	<input type="text" value="Policy Label"/>	
entity-type *	<input type="text" value="hive*"/>	<input checked="" type="radio"/> include
Entity Classification *	<input type="text" value="*"/>	<input checked="" type="radio"/> include
Entity ID *	<input type="text" value="my_db.*"/>	<input checked="" type="radio"/> include
Description	<input type="text" value="Access to metadata entities for Hive database my_db"/>	
Audit Logging	<input checked="" type="radio"/> YES	

**Allow Conditions :**

Select Group	Select User	Add Permissions
<input type="text" value="Select Group"/>	<input type="text" value="admin"/>	

- 管理员操作 遵循授权策略允许用户'admin'执行export/import管理操作。

**Policy Details :**

Policy Type **Access** **Add Valid**

Policy Name *	Admin operations	<b>enabled</b> <input checked="" type="radio"/> <input type="radio"/> normal
Policy Label	Policy Label	
atlas-service *	<input type="text" value="*"/>	<b>include</b> <input checked="" type="radio"/>
Description	Authorization of admin operations in Atlas instance	
Audit Logging	<b>YES</b> <input checked="" type="radio"/>	

**Allow Conditions :**

**add/edit permissions**

Admin Export  
 Admin Import  
 Select/Deselect All

**Add Permissions**

Select Group	Select User	Delegation
<input type="text" value="Select Group"/>	<input type="text" value="admin"/>	<input type="checkbox"/> <input style="border: none; background-color: transparent; font-size: small; margin-right: 10px;" type="button" value="X"/>

## Apache Ranger访问Apache Atlas授权审计

Apache Ranger授权插件生成审核日志，其中包含插件授权的访问权限的详细信息。详细信息包括访问的对象（例如，ID为cost\_savings.claim\_savings@cl1的hive\_table），执行的访问类型（例如，实体添加分类，实体删除分类），用户名，访问时间和IP来自访问请求的地址 - 如下图所示。

Audit Log								
Policy ID	Event Time	User	Service	Resource	Access Type	Result	Access Enforcer	Client IP
Name / Type	Name / Type							
4	05/24/2018 12:29:06 AM	admin	cl1_atlas	hive_table/[INSURANCE, FINAN... entity	entity-remove-classification	<b>Allowed</b>	ranger-acl	10.0.2.2
4	05/24/2018 12:29:06 AM	admin	cl1_atlas	hive_table/[INSURANCE, FINAN... entity	entity-read-classification	<b>Allowed</b>	ranger-acl	10.0.2.2
4	05/24/2018 12:28:37 AM	admin	cl1_atlas	hive_table/[INSURANCE, DATA,... entity	entity-add-classification	<b>Allowed</b>	ranger-acl	10.0.2.2
4	05/24/2018 12:28:37 AM	admin	cl1_atlas	hive_table/[INSURANCE, DATA_QUALITY],FINANCE_PII/cost_savings.claim_savings@cl1 entity	entity-read-classification	<b>Allowed</b>	ranger-acl	10.0.2.2

- 09. 分类传播

- 举个栗子

- 给实体添加分类
    - 更新与实体关联的分类
    - 删除与实体关联的分类
    - 给实体添加血缘
    - 删实体

- 传播控制

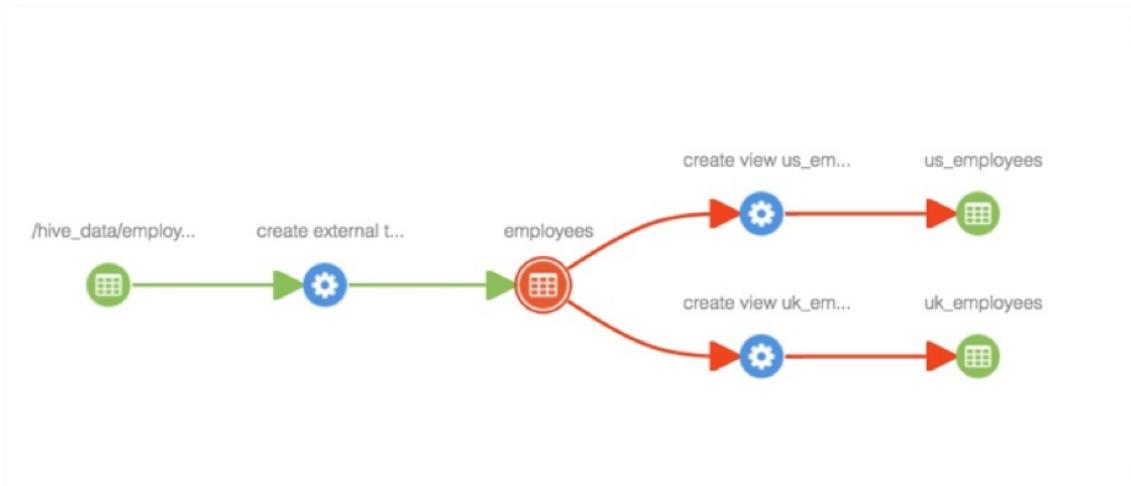
- 分类中的传播标记
    - 血缘流向中的传播标记
    - 在血缘流向中禁用指定分类的传播
    - 通知和审计
    - 术语表

## 09. 分类传播

分类传播使得与实体相关联的分类能够自动与实体的其他相关实体相关联。这在处理数据集从其他数据集中导出数据的场景时非常有用。例如：在文件中加载数据的表，从表/视图生成的报表等。例如，当表被分类为“PII”时，从该表中导出数据的表格或视图（通过CTAS或“创建视图”操作）将自动归类为“PII”。

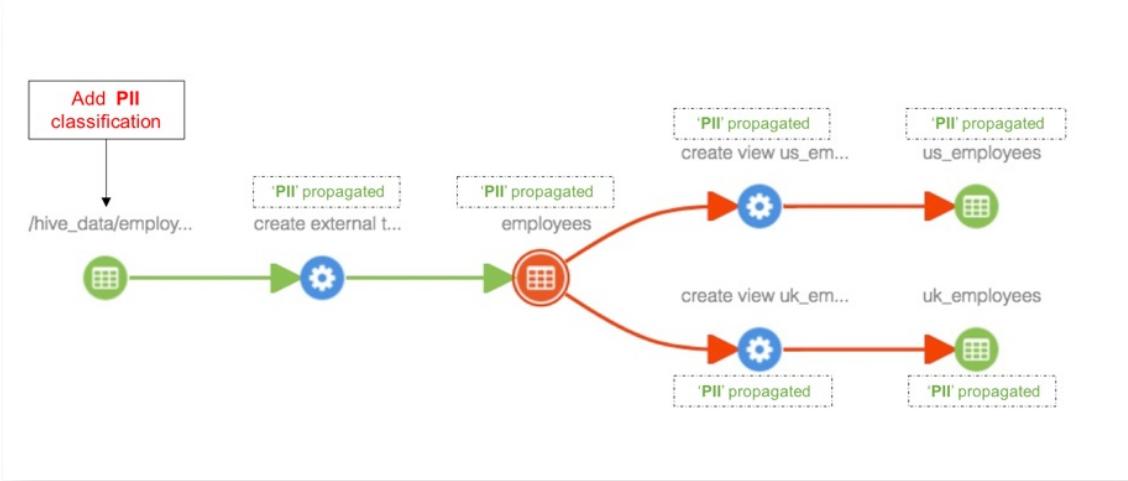
### 举个栗子

假设有以下血缘：将来自 `hdfs_path` 实体的数据加载到表中，该表可通过视图进一步提供。我们将通过各种场景来理解分类传播特性。



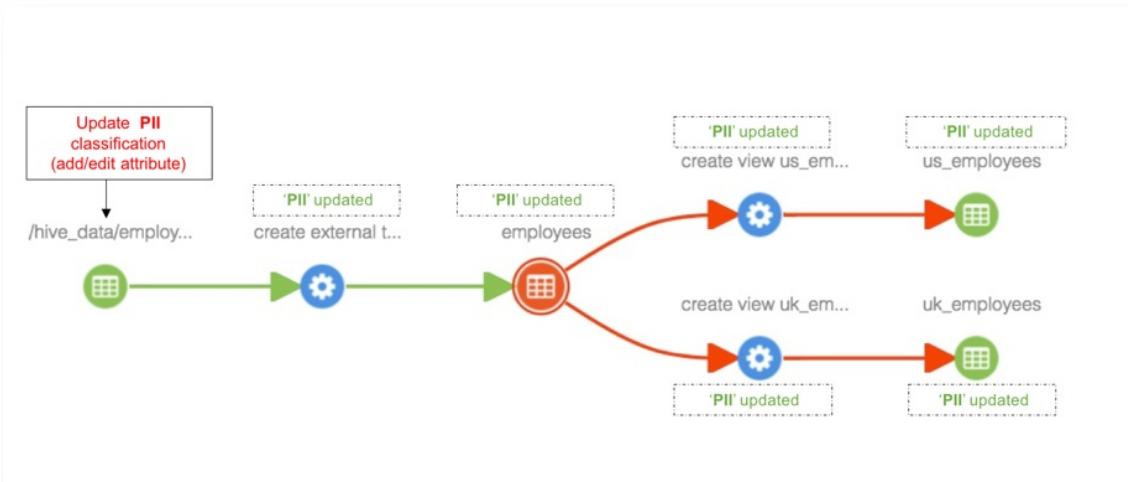
### 给实体添加分类

当分类‘PII’被添加到‘`hdfs_path`’实体时，分类将传播到血缘路径中的所有受影响实体，包括‘`employees`’表，视图‘`us_employees`’和‘`uk_employees`’ - 如下所示。



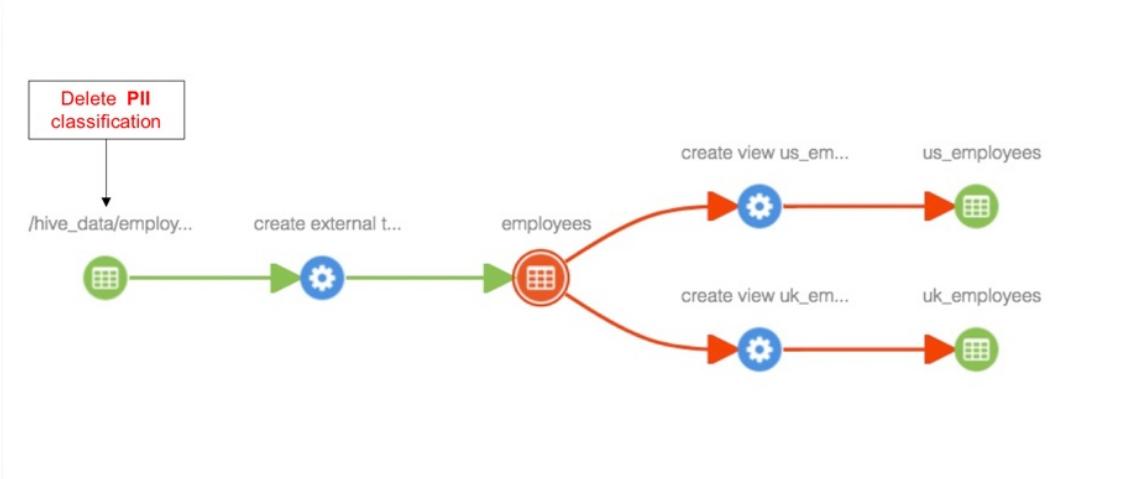
## 更新与实体关联的分类

对于与实体相关联的分类的任何更新都将在分类传播到的所有实体中看到。



## 删除与实体关联的分类

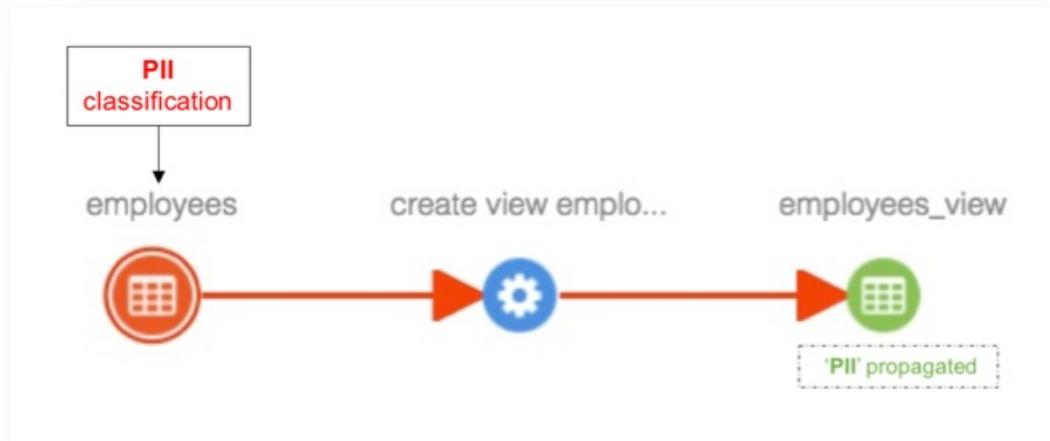
从实体中删除分类时，分类将从分类传播到的所有实体中删除。



## 给实体添加血缘

当在实体之间添加血缘时，例如为了捕获文件中的数据到表的加载，与源实体相关联的分类也被传播到所有受影响的实体。

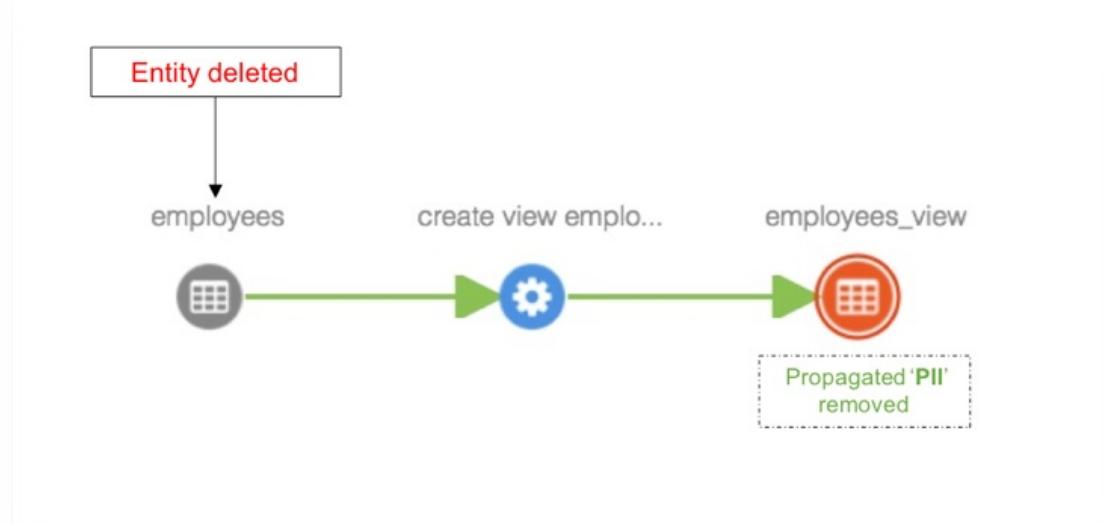
例如，当从表创建视图时，与表关联的分类也会传播到新创建的视图。



## 删除实体

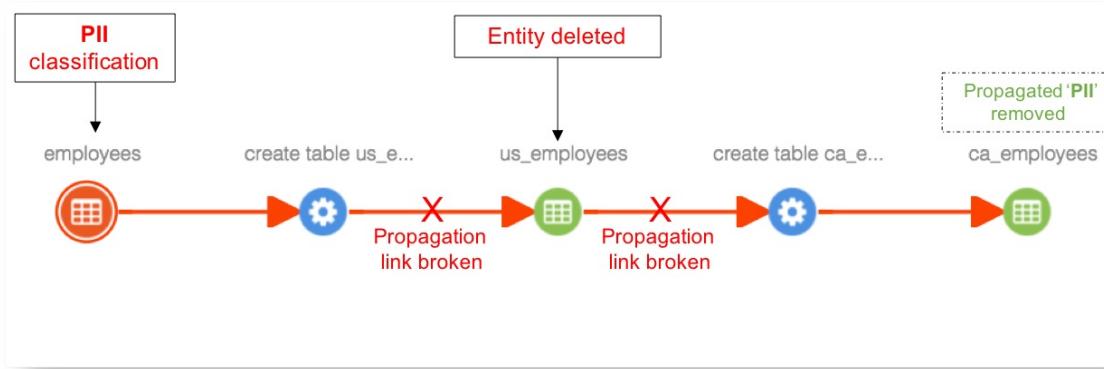
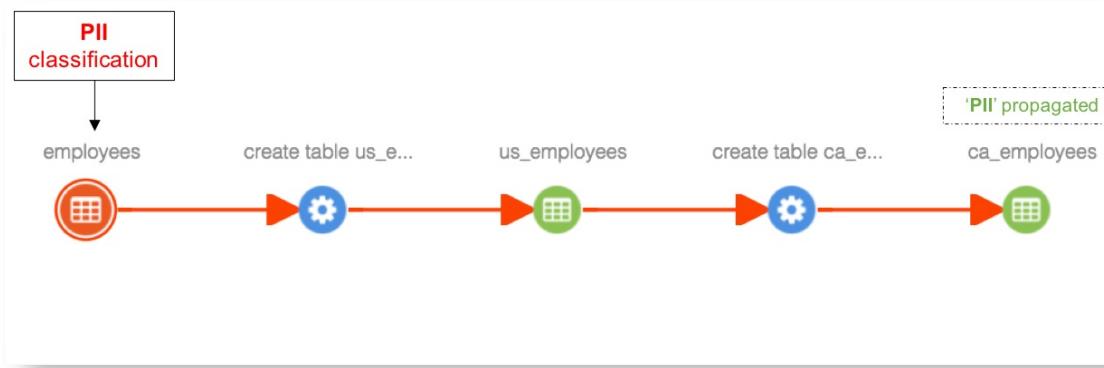
情况1：删除实体时，将从分类传播到的所有实体中删除与此实体关联的分类。

例如。删除employees表时，将从“employees\_view”视图中删除与此表关联的分类。



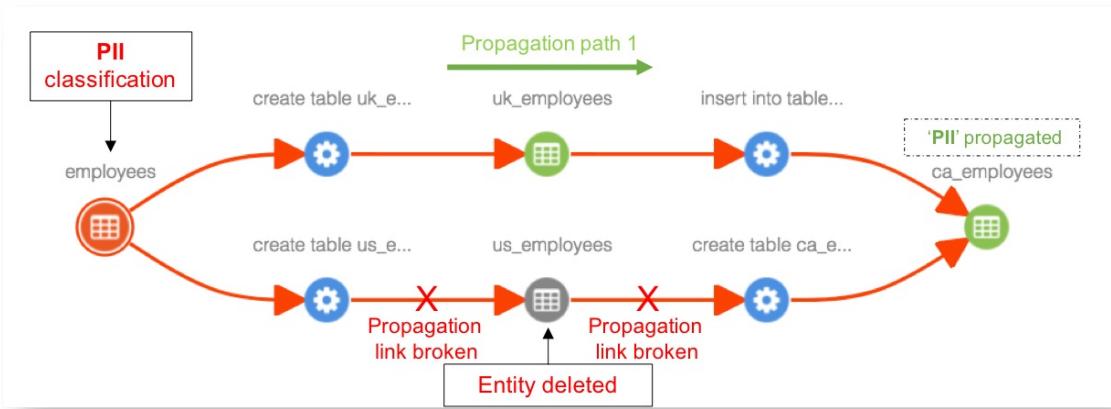
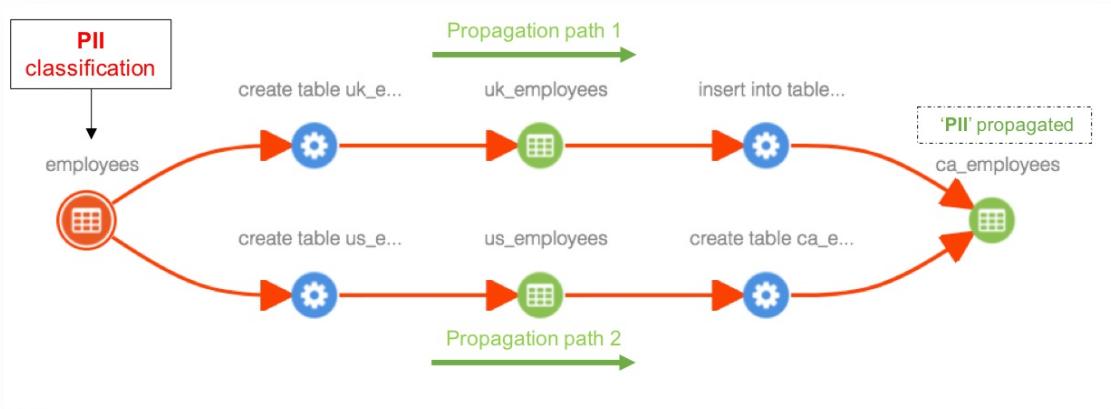
情况2：当在血缘路径的中间删除实体时，传播链接被破坏，并且先前传播的分类将从被删除实体的所有派生实体中移除。

例如。删除'us\_employees'表时，从'ca\_employees'表中删除通过此表（PII）传播的分类，因为实体删除会破坏传播的唯一路径。



情况3：当在血缘路径的中间删除实体并且如果存在用于传播的备用路径时，将保留先前传播的分类。

例如。当'us\_employees'表被删除时，通过该表传播（PII）的分类将保留在'ca\_employees'表中，因为有两个可用的传播路径，其中只有一个被实体删除打破。

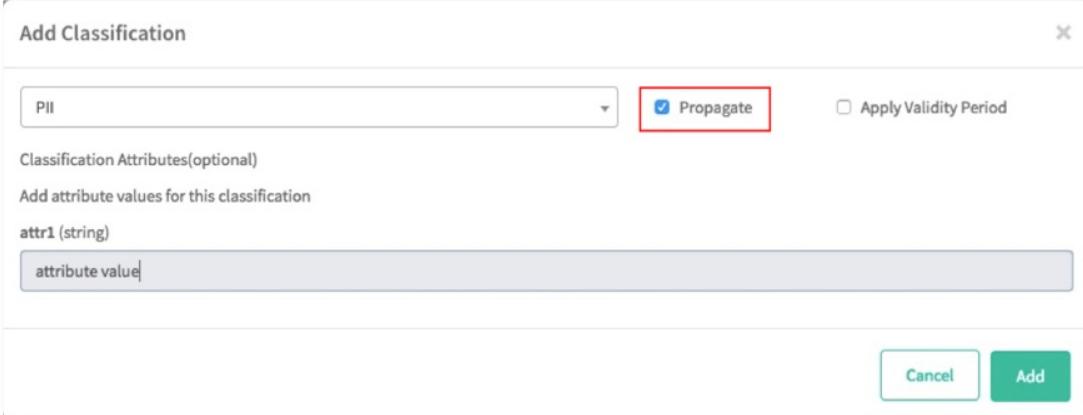


## 传播控制

Apache Atlas提供了一些的选项来控制是否/在何处传播分类。本节将详细介绍这些可用选项。

### 分类中的传播标记

每个分类与实体的关联都有一个布尔标记，用于控制分类是否传播。当分类与实体相关联时，该标记被设置为true，即分类将被传播到所有受影响的实体。在初始关联期间或之后，可以将该标志更新为期望值。



## 血缘流向中的传播标记

Apache Atlas支持在血缘流向中添加标志，以启用/禁用通过流向传播分类。默认情况下，为血缘流向启用传播。当标志关闭时，不会通过此流向传播分类；并且将重新评估当前传播的分类通过流向的传播，以便可以从受影响的实体中移除它们。当标志打开时，将重新评估源实体的分类传播，以便它们可以传播到所有受影响的实体。

## 在血缘流向中禁用指定分类的传播

Apache Atlas支持在血缘流向中禁用特定分类传播。例如，这可以用于处理以下场景：在创建视图时屏蔽分类为PII的列；在这种情况下，如果创建的视图中的相应列可能没有PII，则应阻止PII分类的传播。这可以通过更新血缘流向以在“阻塞的传播分类”列表中添加PII分类来完成。阻塞传播分类中的分类不会在衍生/下游实体中传播。

Block Propagation Table		
Propagation flow <input checked="" type="checkbox"/> Block propagation		
Block Propagation Table		
Classification	Entity Name	Block Propagation
PII	employees (hive_table)	<input type="checkbox"/>
PII	/hive_data/employee (hdfs_path)	<input type="checkbox"/>
VENDOR_PII	employees (hive_table)	<input type="checkbox"/>
SENSITIVE	/hive_data/employee (hdfs_path)	<input type="checkbox"/>

## 通知和审计

添加/更新/删除传播的分类时，Apache Atlas会针对受传播影响的每个实体向“ATLAS\_ENTITIES”主题发送通知。

## 术语表

当分类与术语相关联时，分类将自动传播到与术语相关联的所有实体。



- 12.1 HBase Hook & Bridge
  - HBase 模型
  - HBase Hook
  - 注意
  - 导入HBase元数据

## 12.1 HBase Hook & Bridge

### HBase 模型

HBase模型包含以下类型(type):

- Entity types:
  - hbase\_namespace
    - super-types: !Asset
    - attributes: qualifiedName, name, description, owner, clusterName, parameters, createTime, modifiedTime
  - hbase\_table
    - super-types: DataSet
    - attributes: qualifiedName, name, description, owner, namespace, column\_families, uri, parameters, createtime, modifiedtime, maxfilesize, isReadOnly, isCompactionEnabled, isNormalizationEnabled, ReplicaPerRegion, Durability
  - hbase\_column\_family
    - super-types: DataSet
    - attributes: qualifiedName, name, description, owner, columns, createTime, bloomFilterType, compressionType, compactionCompressionType, encryptionType, inMemoryCompactionPolicy, keepDeletedCells, maxversions, minVersions, datablockEncoding, storagePolicy, ttl, blockCachedEnabled, cacheBloomsOnWrite, cacheDataOnWrite, evictBlocksOnClose, prefetchBlocksOnOpen, newVersionsBehavior, isMobEnabled, mobCompactPartitionPolicy

使用唯一属性qualifiedName在Atlas中创建和删除HBase实体，其值应格式如下所述。请注意，namespaceName, tableName和columnFamilyName应为小写。

```
hbase_namespace.qualifiedName:      <namespaceName>@<clusterName>
hbase_table.qualifiedName:          <namespaceName>:<tableName>@<clusterName>
hbase_column_family.qualifiedName: <namespaceName>:<tableName>.<columnFamilyName>@<clusterName>
```

### HBase Hook

Atlas HBase hook与HBase master注册为协处理器。在检测到对HBase名称空间/表/列族的更改时，Atlas Hook通过Kafka通知更新Atlas中的元数据。按照以下说明在HBase中设置Atlas Hook：

通过添加以下内容在 `hbase-site.xml` 中注册Atlas hook:

```
<property>
  <name>hbase.coprocessor.master.classes</name>
  <value>org.apache.atlas.hbase.hook.HBaseAtlasCoprocessor</value>
</property>
```

- 解压 `apache-atlas-${project.version}-hbase-hook.tar.gz`
- `cd apache-atlas-hbase-hook-${project.version}`

- 将文件夹 `apache-atlas-hbase-hook-${project.version}/hook/hbase` 的全部内容复制到 `<atlas package>/hook/hbase`
- 在HBase类路径中链接Atlas hook jar `ln -s <atlas package>/hook/hbase/* <hbase-home>/lib/`
- 将 `<atlas-conf>/atlas-application.properties` 复制到HBase conf 目录。

`atlas-application.properties` 中的以下属性控制线程池和通知详细信息：

```
atlas.hook.hbase.synchronous=false # whether to run the hook synchronously. false recommended to avoid delays in HBase operations. Default: false
atlas.hook.hbase.numRetries=3      # number of retries for notification failure. Default: 3
atlas.hook.hbase.queueSize=10000   # queue size for the threadpool. Default: 10000

atlas.cluster.name=primary # clusterName to use in qualifiedName of entities. Default: primary

atlas.kafka.zookeeper.connect=          # Zookeeper connect URL for Kafka. Example: localhost:2181
atlas.kafka.zookeeper.connection.timeout.ms=30000 # Zookeeper connection timeout. Default: 30000
atlas.kafka.zookeeper.session.timeout.ms=60000  # Zookeeper session timeout. Default: 60000
atlas.kafka.zookeeper.sync.time.ms=20     # Zookeeper sync time. Default: 20
```

可以通过在配置名称前加上“`atlas.kafka`”前缀来指定Kafka通知生成器的其他配置。有关Kafka制作人支持的配置列表，请参阅[Kafka Producer Configs](#)

## 注意

Atlas HBase hook仅捕获名称空间、表和列族的创建/更新/删除操作。将捕获对列的更改。

## 导入HBase元数据

Apache Atlas提供了一个命令行实用程序 `import-hbase.sh`，用于将Apache HBase名称空间和表的元数据导入Apache Atlas。此实用程序可用于使用Apache HBase集群中存在的名称空间/表初始化Apache Atlas。此实用程序支持导入特定表的元数据，特定命名空间中的表或所有表。

```
Usage 1: <atlas package>/hook-bin/import-hbase.sh
Usage 2: <atlas package>/hook-bin/import-hbase.sh [-n <namespace regex> OR --namespace <namespace regex>] [-t <table regex> OR --table <table regex>]
Usage 3: <atlas package>/hook-bin/import-hbase.sh [-f <filename>]
        File Format:
        namespace1:tbl1
        namespace1:tbl2
        namespace2:tbl1
```