# 系统设计

本系统主要基于开源项目Openstack的高可用分布式对象存储服务Swift进行开发设计，Swift作为本系统运行使用时依托的云存储平台，主要有三个组成部分：Proxy Server、Storage Server和Consistency Server，如图 1所示，其中Storage和Consistency服务均允许在Storage Node上，Auth认证和Proxy中间代理则根据本系统的设计方案进行替换实现，以满足多模式访问控制的需求。



图 1 Swift部署架构

## 相关定义与性质

多模式的访问控制主要是协同使用基于身份的访问控制（IBAC）、基于静态属性的访问控制（ABAC）和基于动态属性的访问控制（DABAC），以此实现在不同的应用场景下，结合使用最优的访问控制策略，以实现云环境下多元化的访问控制需求。

**定义 1.1： 用户标识User Id**。用户标识是用于唯一标识用户身份的信息，由系统后台自动生成。

在IBAC中，User Id将作为判断用户合法权限的重要因子。

**定义1.2：属性Attribute（简称attr）**。属性是用户的特征描述，一般是由属性名和属性值构成的二元组成。属性又分为静态属性和动态属性。

**静态属性Static Attr。**静态属性是指属性值相对固定或较少变动的一类属性。属性值固定不变的如性别、相对稳定的如工作及较少变动的年龄等都划分与静态属性的范畴。

**动态属性Dynamic Attr。**动态属性是指属性值为非量化或变动频繁的一类属性。如非量化属性技能熟练度、变动频繁的活跃度等则被划分为动态属性。

说明：静态属性和动态属性的划分并未有严格意义的界限，可以根据具体的需求进行区。

**定义1.3：访问策略Policy。**访问策略是指访问域的安全要求，是访问控制的权限判断所依据的规则。

本系统设计了一种适用于多模式的访问控制模型的访问策略描述规则（DACML），以此实现IBAC、ABAC和DABAC三种不同访问策略的统一描述和规划使用。图 2给出一个具体的Policy示例。

**method标签：**主要用于选择采用的具体访问策略，其值为ABAC，ABAC，DABAC；

**white标签：**表示白名单，在白名单的用户可以直接访问数据，即便其不满足访问策略规则；

**black标签**：表示黑名单，黑名单中的用户无权访问数据；

**policy标签：**具体的访问策略逻辑表达式。

**rule标签：**访问策略的逻辑子单元，其中item条目通常为属性应满足的条件表达式

DACML的具体使用规则如下：

1. 首先判断用户的连接请求，读取对应的method标签值，确定访问策略方案
2. 判断用户是否在white和black标签值中，若存在，则可以直接判断用户的访问权限；否则在非IBAC访问策略下继续执行步骤3，若是IBAC则直接拒绝用户访问请求
3. 读取policy标签逻辑表达式，判断用户的属性集是否满足policy标签的逻辑表达式，若不满足表示用户无权访问；否则表示用户拥有合法的访问权限

注：在DABAC中，policy中的item值通常为动态属性，进行逻辑表达式匹配时将不仅仅是简单的是或否进行决定，具体可以参看.。。。

|  |
| --- |
| <DACML>  <method value=**"DABAC"** /> <!-- choose one in [ ABAC | ABAC | DABAC ] -->  <white list=**"Bob, Jim, Lucy"** />  <black list=**"Lily, Sally"** />  <rule>  <item name=**"item1"** attr=**"programming language"** value=**"one in {android, object-c, c#}"** rate=1/>  <item name=**"item2"** attr=**"job"** value=**"client developer"** rate=1 />  <item name=**"item3"** attr=**"skill"** value=**"junior"** rate=1 />  <item name=**"item4"** attr=**"work years"** value=**">3"** rate=2 />  <item name=**"item5"** attr=**"os"** value=**"linux"** rate=1 />  <item name=**"item6"** attr=**"job"** value=**"server developer"** rate=1 />  </rule>  <policy  value=**"(rate>2 in {item1, item2, item3}) or (rate>2 in {item4, item5, item6})"** >  </policy>  </DACML> |

图 2 访问策略描述示例

**定义1.4 访问控制（Access Control）。**按用户身份及其所归属的某项定义组来限制用户对某些信息项的访问，或限制对某些控制功能的使用的一种技术。

**定义1.5 授权令牌（Auth Token）。**授权令牌是指系统中一种表征用户身份的数据信息结构。Auth Token是用户经过登录认证后，由系统返回，且具有一定的生命周期。用户在与系统进行数据操作时，通过发送Auth Token表示用户在系统中的合法身份。本文中授权令牌主要分为Basic Auth Token和New Auth Token（or Extend Auth Token）

**Basic Auth Token：**指由登录系统后返回的原始Auth Token

**New Auth Token：**指在Basic Auth Token的基础上进行扩充，包含用户的静态属性集或动态属性集，以此应用于系统中用户权限判断的Auth Token

**定义1.6 域（ Area）。**域是指系统中存储数据区域的逻辑划分。域可分为个人域（Secret Area，简称SA）、群组域（Group Area，简称GA）和公共域（public Area，简称PA）。

**个人域（SA）：**又指用户私人空间，用于存放用户个人数据的存储区域，通常一个用户（User）对应一个SA；

**群组域（GA）：**又指组空间，用于存放组内数据共享的存储区域。通常一个组（Group）对应一个GA；

**公共域（PA）：**又指公共空间，用于存放面向所有用户数据共享的存储区域。

## 系统存储分布

本系统的存储结构分布如图 3所示，其中Proxy Server作为用户同swift存储平台进行数据交互的接口，接受用户发出的请求，并根据请求的类型做出不同的决策从而跳转到存储端Storage Node的相关部分执行数据操作请求。Authentication Server主要是在用户发起登录请求时，判断用户的合法身份，若合法则生成Auth Token并缓存到Cache Server中直至过期时间，同时将Auth Token返回给用户，作为用户登录后操作请求的身份凭证，相应的后续操作中用户的Auth Token判断在告诉缓存Cache Server中执行。此外Cache Server同样缓存账户和容器的存在信息，但不会缓存对象本身的数据，以此提高用户身份判断、数据操作请求的效率。



图 3 系统存储分布图

本系统中依旧采用swift存储结构中的Ring来实现数据定位，与之不同的则是在定位到具体的数据之后，执行数据访问之前，将执行访问权限判断。本系统采用全新的多模式访问控制方法替换swift自带访问控制列表，两者对比可以参见表 1。相较于swift简单的访问控制而言，本系统的多模式访问控制系统更加适用于云存储中多用户环境，在用户基数规模较大的前提下，采用ABAC或DABAC的存储开销将远小于IBAC的存储开销，具体可以参考测试章节；此外ABAC或DABAC的访问策略制定也将远优于IBAC的合法权限用户的挑选。

表 1 访问控制访问对比

|  |  |  |
| --- | --- | --- |
|  | **Swift访问控制方法** | **多模式访问控制方法** |
| 访问控制方法 | 仅支持IBAC | 支持IBAC、ABAC、DABAC |
| 扩展性 | 差 | 强 |
| 自适应性 | 弱 | 强，可根据需要满足多种环境 |
| 存储开销 | 与用户数量成正比 | 与Policy复杂度相关，变化幅度小 |
| 性能开销 | 与用户基数成正比 | 与Policy复杂度相关 |
| 粒度 | 细粒度 | 根据需要可选择细粒度和粗粒度 |

本系统中Storage Server提供数据存储服务，其数据模型如图 4所示，根据具体的存储服务可分为Account、Container和Object存储服务器。其中Account服务器负责保存用户的个人账号信息，群组账号信息，公共账号信息并维护账号下所包含的容器列表服务；Container Server提供容器元数据和统计信息，并维护所含有对象列表的服务；Object Server则实现具体的数据存储，并提供对象元数据和相关附属信息。



图 4 数据模型

从图 3中可以看出，用户的动态属性集保存在个人账号之下，一个用户将只维护一张Dynamic Attr Table，且一个用户也将只有一个Private Account；一个Group Area对应一个Group Account，用户的组内属性（也即Static Attr）保存在相应的Group Account下，一个用户可以加入多个群，每个群的组内属性也可以不同，且只有群内成员才能访问群内数据；Public Area对应Public Account，所有用户都可以获得Public Account下的数据信息，但涉及到具体的Object访问时需要进行权限判断，当满足Object的Access Policy时方能访问。

Container是swift提供的一个类似文件系统中的文件夹或目录的概念。Container服务器维护挂载在其下的Object列表信息，且Container同样支持访问权限的设置，从而实现访问控制分层的概念。具体而言，Container将维护一个上层的访问策略，表示该Container下的所有Object的访问前提是用户满足Container的访问策略，再次基础之上，在判断用户是否满足Object的访问策略。

举例如下：

在IT群类，有一个容器Image Resource，容器下有诸多图片，访问策略为

表示只有developer或editor用户才能访问Image Resource下的数据，且只有Coco和Bob两个人能访问image1.jpg，其他的用户即便是developer或editor也不能访问image1.jpg

Object服务器存储用户上传的数据及其相关的文件信息，其中文件附属信息包括制定的访问策略、TAG标签、owner，创建时间、修改时间等。访问策略作为object的附属信息保存，因此每个object可以制定一个独立的访问策略，从结构上来讲，不同域的object无任何区别，只是对应的访问策略不同，且一般采用DABAC访问控制方法的object将带有TAG标签，用于用户动态属性的校正，就具体可以参看。。。

## 系统结构

本系统主要设计一种适用于云环境的多模式访问控制协同工作方案，其中，针对云环境中不同的应用场景，采用各自适用的访问控制方案。本系统中云环境主要划分为三个区域：个人域（secret area）、群组域（group area）和公共域（public area）。如图 5 给出系统结构图。



图 5 系统结构图

从系统结构图中可以看出，本系统主要分为认证、访问策略制定、访问控制方法决策以及用户授权。其中认证主要实现用户登录认证，并对合法用户返回 Auth Token以作为用户后续访问操作的身份凭证；访问策略制定与封装包括用户选择合适的访问控制方法，按照要求制定相应的访问策略，并实现对用户制定访问策略的统一封装，以此对外实现统一的调用接口；访问控制方法决策则表示Proxy Server解析用户访问数据的访问策略，以此判定具体采用的访问控制方法并实现访问权限判断的分流；用户授权则主要是判断用户的权限并给出相应的相应请求。

### 认证

认证应用于用户登录系统之时，通过判断用户登录请求中的有效信息是否合法，从而决定是否授予用户Auth Token。

用户首先通过用户名+口令的方式请求登录系统，在Authorize Server处检测用户登录请求信息的合法性，若用户信息合法，则生成一个唯一的Auth Token并缓存到Cache Server中，同时将生成的Auth Token返回给用户。

用户将持有Auth Token进行后续的操作请求，存储端在Cache Server中进行判断用户的Auth Token是否合法。由于Auth Token区分为Basic Auth Token和New Auth Token，其结构如图 6， 其中后者主要应用于采用ABAC和DABAC的访问控制方法的object访问中，因此对于New Auth Token而言，首先将在Cache Server中进行结构解析，得到Basic Auth Token然后判断其是否合法（合法指Auth Token仍在生命周期内，且cache内有记录）。



图 6 Auth Token结构图

### 访问策略制定

访问策略的制定是本系统的一个关键点，与其他访问控制方法的不同点之一在于我们采用的访问策略的极大差异性。 定义1.3给出了访问策略的一般性描述，并对本系统设计的访问策略结构进行了描述和示例说明。从图5中也可以看出，访问策略制定需要结合具体采用的访问方法，在不同的访问域内，访问策略的制定并不完全相同。图 7显示了policy的结构图



图 7 policy结构图

IBAC访问方法的策略制定主要在于选定可以访问的用户列表或禁止访问列表，即制定white list和black list，将用户以列表形式保存在policy中；

ABAC访问方法的Policy是二叉树的结构，其中Policy Tree的叶子节点为静态属性，非叶子节点则为and、or、not等逻辑词；

DABAC访问方法的policy同样是Tree的结构，不同与ABAC的是DABAC的Policy Tree通常是多叉树，其叶子节点为动态属性应满足的条件，非叶子节点则为一个权重值、or及and等。

### 权限判断及用户授权

用户权限判断是访问控制中至关重要的一个环节，权限判断的结果将直接表明用户受否拥有访问数据的权限，同时由于权限判断与访问策略相关，根据上一节可以看出，本系统的访问权限判断同样在不同的访问方法下，有不同的匹配算法。本节将以一个完整的访问流程来具体说明访问权限判断及授权的工作原理， 是一个完整的访问流程。



图 8 访问流程

1. Init初始化

选中文件，并制定相应的访问策略，上传到云存储空间。在不同的Access Area上传数据，且为有访问权限设定的数据制定访问策略

1. 用户登录系统

用户以用户名+密码的方式登录系统，接受并缓存返回的Auth Token

1. 访问请求

用户发出访问请求，存储端Proxy Server接受并处理用户请求，通过ring环定位并转到具体的访问对象，然后执行访问权限判断，若权限判断通过则响应用户请求否则返回相应的错误码。

**权限判断：**

（2-1）IBAC：访问控制列表的结构，因此需要判断用户的User ID是否在white list或black list中，以此实现用户权限判断

（2-2）ABAC：二叉访问树结构，这里主要运用到用户的静态属性集与访问树之间的匹配算法，如表 2所示，首先根据制定的policy构建二叉树，其中二叉树的所有叶子节点为属性，非叶子节点为and、or和not等逻辑词；匹配算法则根据叶子节点与用户静态属性集之间的映射关系决定，and逻辑词要求左右孩子节点都满足映射关系，or逻辑词要求左右孩子节点至少有一个满足而not逻辑词则要求不满足，根据以上原则，递归到根节点，得出判断结果。

表 2 ABAC权限判断算法

|  |  |
| --- | --- |
| **Algoritm1 generateTree( rule )** | **Algoritm2 match(node, attres)** |
| **step1** : /\*初始化两个堆栈\*/  InitStack(string\_stack); /\*存放符号的堆栈\*/  InitStack(node\_stack); /\*存放树节点的堆栈\*/  **step2** : /\*对输入的字符串进行扫描\*/  **while**(attr = get(Policy)) // 获得属性  **switch**(attr) {  **case** ‘(’: 将‘(’压入string\_stack; break;  **case** ‘)’: 以string\_stack栈顶元素为值创建一个二叉树结点;  弹出string\_stack栈顶元素;  二叉树结点的左右指针指向node\_stack栈顶的两个结点;  弹出string\_stack栈顶的两个节点;  break;  **case** ‘and’: 将‘and’压入string\_stack; break;  **case** ‘or’: 将‘or’压入string\_stack; break;  **default**: 以该串为值创建一个二叉树结点，并压入node\_stack; break;  }  **Step3**:  以string\_stack栈顶元素为值创建一个二叉树结点root  其左右孩子分别指向node\_stack栈顶的两个元素  return root; | **if**(root -> value == "not"){  **return** match(root->lchild, attrs) ? 0 : 1;  } **else if**( root -> value == "and"){  **return** (match(root->lchild, attrs) +  match(root->rchild, attrs)) == 2 ? 1 : 0;  } **else** **if**( root -> value == "or"){  **if**(match(root->lchild, attrs) ||  match(root->rchild, attrs))  **return** 1;  **else**  **return** 0;  }**else** {  **if** ( node -> value **in** attres) // 叶子属性在用户属性集内，返回1  **return** 1;  else  **return** 0;  } |

（2-3）DABAC：多叉树结构