Hadoop授权机制分析

Hadoop安全机制由认证、授权、审计等构成：认证是对一个实体的身份进行判断，确保用户是自己声称的用户；授权是系统管理员授予实体用户是否能够访问某资源的权利，其也是系统安全的基本需求，认证和授权机制之间相互配合完成Hadoop的安全运行。授权机制，如下图所示：



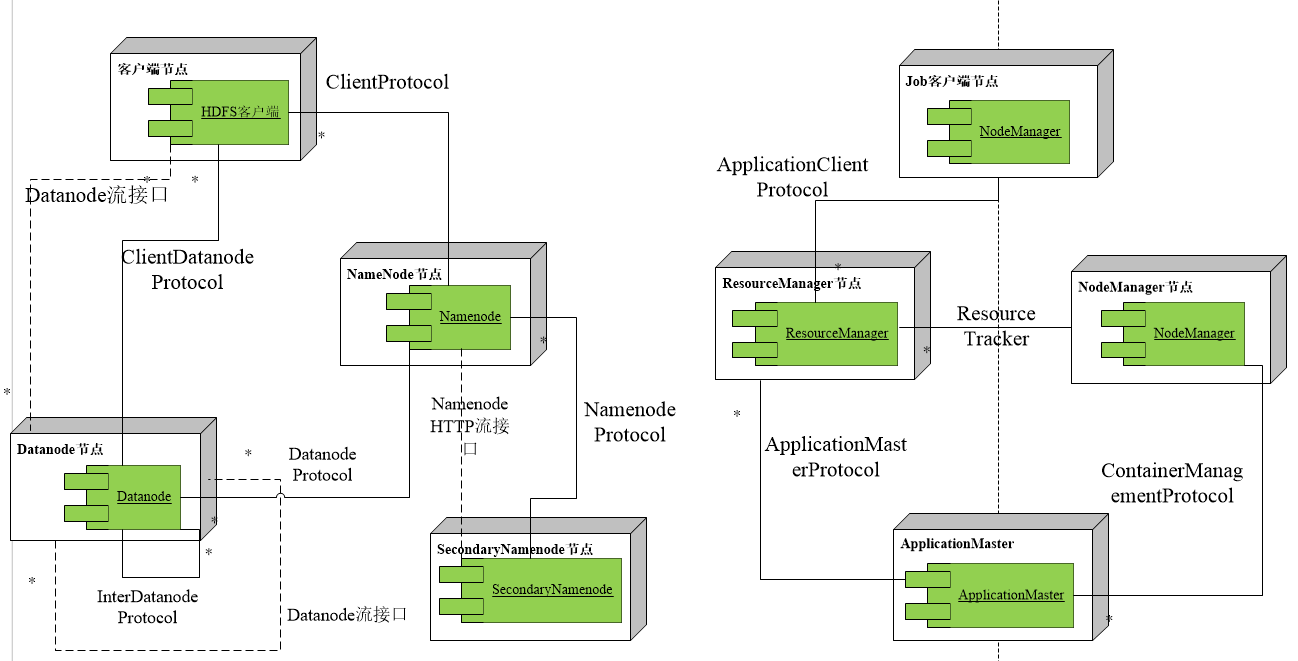
Hadoop的授权分为两个层次及第三方访问控制体系

1. **SLA(Service Level Authorization)**，用于控制用户是否可访问指定的服务，其通过验证用户是否可以通过某种Protocol协议来访问特定用户，例如用户通过hdfs#ClientProtocol来访问NameNode，这不涉及到具体的操作。SLA是最基础的访问控制，其优先于文件权限、队列的权限控制等
2. **资源权限控制**，这涉及到具体的资源的访问权限，包括用户是否有权限访问HDFS中某个目录、是否可以向某个队列中提交Job、是否有控制权限来管理启动的应用等。在这一层次中文件、队列、应用等都可以作为资源，对某类资源的权限控制有相应的控制模块，例如DFS Permission模块用于文件权限的管理。
3. **Third Party Authorization Provider**，Hadoop中资源的权限控制有相应模块进行管理，其控制策略不同而且分散在不同模块中很难去维护，这就需要集中式安全管理框架，目前主流的第三方安全管理框架有Apache Ranger和Sentry。集中式安全管理框架针对各种资源提供了统一的访问权限模型和管理界面，使集群管理员可以轻松的通过配置策略来控制各资源的访问权限。

下面对各层次的控制体系进行详细介绍。

# Service-Level Authorization

在Hadoop中，客户端和各组件之间的交互都是通过RPC Protocol来实现，如下图：

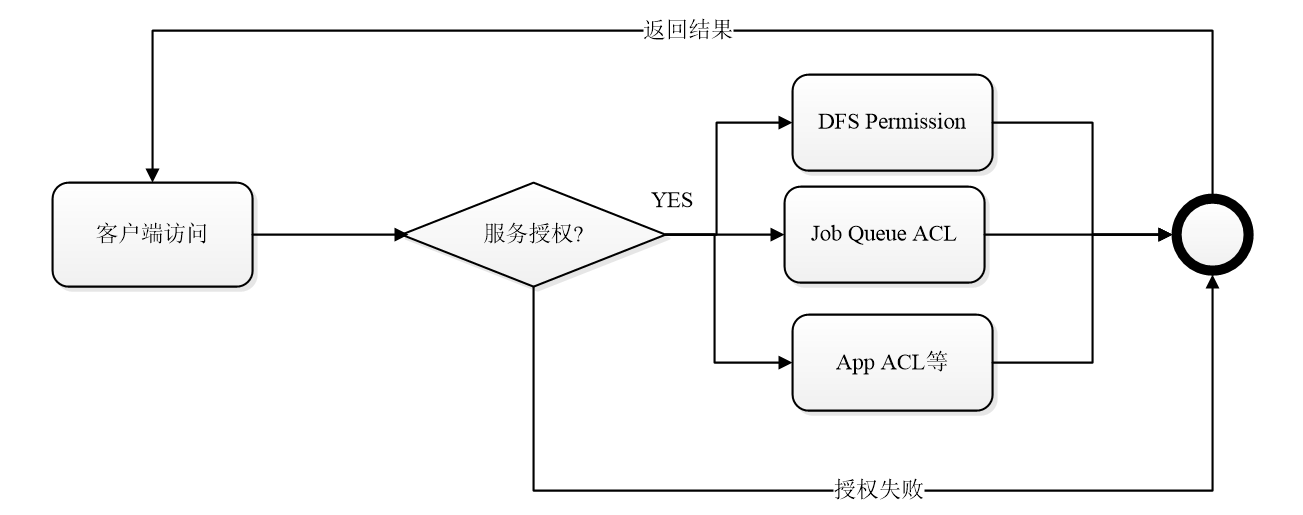


Hadoop采用了Pull-based通信模型，客户端获取信息要主动的通过RPC协议与Server端进行交互，具体各个Protocol提供的接口不再详细介绍。使用SLA可以实现精确的服务访问控制：

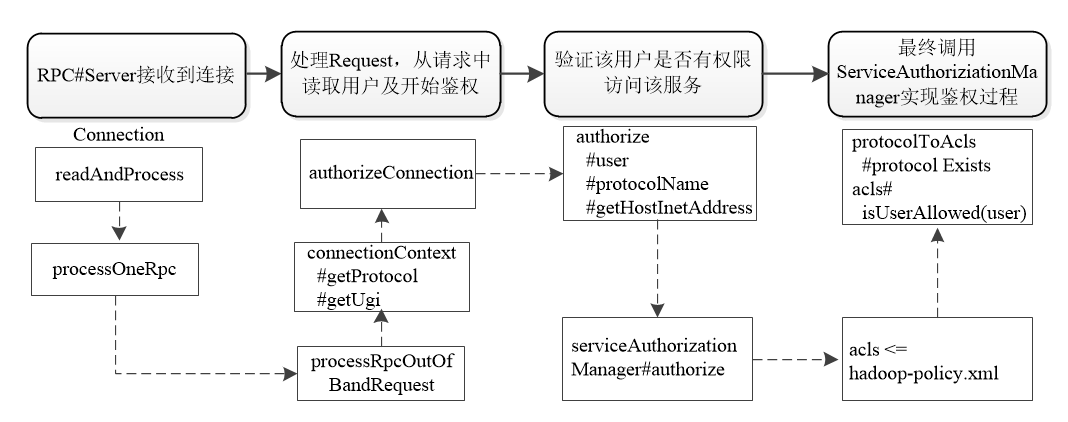
* 限制某些用户访问给定服务
* 限制某主机访问给定服务
* 拒绝客户端访问服务中不支持的协议

## 1.1 SLA授权流程

服务授权（Service-Level Authorization）是最基础的授权机制保证客户端能够有权限访问给定的服务，在Hadoop访问流程中服务授权优先于文件权限、Job Queue ACL等，其流程如下：



Ipc#Server接收到客户端请求后，首先进行服务授权，其核心执行如下：



ServiceAuthorizationManager#authorize的执行源码如下所示：

*public void authorize(UserGroupInformation user,*

*Class<?> protocol,*

*Configuration conf,*

*InetAddress addr*

*) throws AuthorizationException {*

*AccessControlList[] acls = protocolToAcls.get(protocol);*

*MachineList[] hosts = protocolToMachineLists.get(protocol);*

*//验证该组件是否支持客户端使用的Protocol，访问协议*

*if (acls == null || hosts == null) {*

*throw new AuthorizationException("Protocol " + protocol +*

*" is not known.");*

*}*

*// 开启Krb认证后，从配置中获取该protocol 授权的principal*

*clientPrincipal = SecurityUtil.getServerPrincipal(conf.get(clientKey), addr);*

*//验证客户端使用的用户是否属于授权用户*

*//1.用户是否为配置中clientPrincipal*

*//2.acl列表中有该用户*

*if((clientPrincipal != null && !clientPrincipal.equals(user.getUserName())) ||*

*acls.length != 2 || !acls[0].isUserAllowed(user) || acls[1].isUserAllowed(user)) {*

*}*

*//policy.xml中配置了该协议所允许的访问主机，则验证该host是否属于配置主机列表*

*if (addr != null) {*

*String hostAddress = addr.getHostAddress();*

*if (hosts.length != 2 || !hosts[0].includes(hostAddress) ||*

*hosts[1].includes(hostAddress)) {*

*}} }*

其验证某个用户是否可以使用服务（即使用的服务Protocol），基于以下条件：

1. 该服务支持用户使用的Protocol，该Protocol注册到服务中，下面会进行介绍注册过程
2. 该用户是否是合法使用用户
   1. 开启krb时，用户是否是配置的clientPrincipal
   2. 用户是否在acl类表中

3）服务端配置其所允许的客户端主机时，验证客户端主机是否在运行的主机列表中

在上面的流程中权限的验证基于服务使用的ACL列表，Hadoop服务的使用acl列表配置在hadoop-policy.xml中，见加载代码：

*String policyFile = // HADOOP\_POLICY\_FILE <=hadoop-policy.xml*

*System.getProperty("hadoop.policy.file", HADOOP\_POLICY\_FILE);*

*// Make a copy of the original config, and load the policy file*

*Configuration policyConf = new Configuration(conf);*

*policyConf.addResource(policyFile);*

*refreshWithLoadedConfiguration(policyConf, provider);*

## **1.2服务Protocol的ACL列表**

hadoop-policy.xml中可配置的参数如下表：

|  |  |  |
| --- | --- | --- |
| 配置项 | 协议名称 | 说明 |
| security.client.protocol.acl | ClientProtocol | 用户代码基于DistributedFileSystem与NameNode交互，可以操作Hadoop的Namespace，以及打开/关闭文件流操作。 |
| security.client.datanode.protocol.acl | ClientDatanodeProtocol | 客户端与DataNode交互协议，用来实现数据库恢复（Block Recovery）。 |
| security.datanode.protocol.acl | DatanodeProtocol | DataNode与NameNode通信的协议，DataNode基于此协议向NameNode发送block report，以及DataNode当前状态信息（如负载情况）。 |
| security.inter.datanode.protocol.acl | InterDatanodeProtocol | DataNode之间进行通信的协议，用来更新Block副本（replica）信息，如时间戳、长度等信息。 |
| security.namenode.protocol.acl | NamenodeProtocol | SecondaryNameNode与NameNode进行通信的协议，用来获取NameNode的状态信息，如进行checkpoint的edits与fsimage。 |
| security.admin.operations.protocol.acl | AdminOperationsProtocol | HDFS管理操作协议。 |
| security.refresh.user.mappings.protocol.acl | RefreshUserMappingsProtocol | 用来刷新缓存中用户与用户组映射关系信息，因为无论是操作HDFS，还是运行MapReduce Job，都会用到用户信息。 |
| security.refresh.policy.protocol.acl | RefreshAuthorizationPolicyProtocol | 用来更新认证策略（Authorization Policy）配置，对应于配置文件/etc/hadoop/hadoop-policy.xml，控制执行hdfs dfsadmin -refreshServiceAcl和yarn rmadmin -refreshServiceAcl的权限。 |
| security.ha.service.protocol.acl | HAServiceProtocol | HDFS HA操作协议，用来管理Active NameNode与Stand-by NameNode状态。 |
| security.zkfc.protocol.acl | ZKFailoverController | ZooKeeper Failover控制器操作权限，用于HDFS HA。 |
| security.qjournal.service.protocol.acl | QJournalProtocol | QuorumJournalManager与JournalNode之间通信的协议，用于HDFS HA，用来同步edits，并协调Active NameNode与Stand-by NameNode状态。 |
| security.mrhs.client.protocol.acl | HSClientProtocol | 客户端与MR History Server之间通讯的协议，用来查看Job历史信息。 |
| security.resourcetracker.protocol.acl | ResourceTracker | ResourceManager与NodeManager之间通信的协议。 |
| security.resourcemanager-administration.protocol.acl | ResourceManagerAdministrationProtocol | ResourceManager管理操作协议。 |
| security.applicationclient.protocol.acl | ApplicationClientProtocol | YARN客户端（Application）与ResourceManager通信的协议，包括Job提交、Job取消，查询Application状态信息等。 |
| security.applicationmaster.protocol.acl | ApplicationMasterProtocol | ApplicationMaster与ResourceManager之间进行通信的协议，包括AM向RM发送注册或注销请求（获取或释放资源）。 |
| security.containermanagement.protocol.acl | ContainerManagementProtocol | ApplicationMaster与NodeManager之间进行通信的协议，包括启动/停止Container调用请求。 |
| security.resourcelocalizer.protocol.acl | LocalizationProtocol | NodeManager与ResourceLocalizer之间进行通信的协议。 |
| security.job.task.protocol.acl | TaskUmbilicalProtocol | Map/Reduce Task进程与后台父进程（向MR ApplicationMaster请求创建Map/Reduce Task）之间进行通信的协议。 |
| security.job.client.protocol.acl | MRClientProtocol | MR JobClient与MR ApplicationMaster之间进行通信的协议，包括查询Job状态等。 |
| security.applicationhistory.protocol.acl | ApplicationHistoryProtocol | 客户端访问timeline Server获取日志 |

以上类表中是ACL相关的配置参数，还可以通过IP、HostName及IP段来进行ACL控制，其定义了可以访问某服务的固定主机信息，其相关配置方式在将上表中的”acl”参数改变成hosts，如下：

*security.client.protocol.acl ->  security.client.protocol.hosts*

其他参数类似，其默认值是“\*”允许所有主机访问。

## **1.3 SLA使用示例**

使用SLA首先在Hadoop配置文件core-site.xml中增加下面的配置

*<property>*

*<name>hadoop.security.authorization</name>*

*<value>true</value>*

*</property>*

该配置属性默认是false，用户可以访问所有地方服务，不进行服务级别的权限认证。

1. ACL 列表，配置SLA权限，实际上增加ACL(配置用户或用户组)，基本格式要求：

* 配置用户和用户组，内容格式：user1,user2 group1,group2，用户和用户组之间使用空格间隔
* 如果只有用户组，配置内容前面增加一个空格” group1,group2”
* 配置内容为“\*”，表示所有用户都具有对应的服务操作权限

1. 配置示例

* 只有hadoop用户具有修改SLA认证权限的配置
* 用户yarn具有向YARN上运行MapReduce Job的权限
* 用户hdfs具有使用DFSClient来访问HDFS

1. 配置示例如下：

*<property>*

*<name>security.job.client.protocol.acl</name>*

*<value>yarn</value>*

*</property>*

*<property>*

*<name>security.refresh.policy.protocol.acl</name>*

*<value>hadoop</value>*

*</property>*

*<property>*

*<name>security.client.protocol.acl</name>*

*<value>hdfs</value>*

*</property>*

1. 配置后，进行下来操作

* 只能通过hadoop用户更改ACL配置，测试如下：

*[hdfs@cmhhost3 ~]$ hdfs dfsadmin -refreshServiceAcl*

*refreshServiceAcl: User hdfs (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.security.authorize.RefreshAuthorizationPolicyProtocol: denied by configured ACL*

hadoop用户执行如下：

*[hadoop@cmhhost3 ~]$ hdfs dfsadmin -refreshServiceAcl*

*Refresh service acl successful*

* hdfs用户访问HDFS

*[mapred@cmhhost3 ~]$ hdfs dfs -ls /*

*ls: User mapred (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.hdfs.protocol.ClientProtocol: denied by configured ACL*

*[yarn@cmhhost3 ~]$ hdfs dfs -ls /*

*ls: User yarn (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.hdfs.protocol.ClientProtocol: denied by configured ACL*

*[hdfs@cmhhost3 ~]$ hdfs dfs -ls /*

*Found 7 items*

*drwxrwxrwt - yarn hadoop 0 2018-11-13 09:39 /app-logs*

*drwxr-xr-x - yarn hadoop 0 2018-11-13 09:30 /ats*

* 只有yarn用户可以提交程序

*[hdfs@cmhhost3 ~]$ hadoop jar /usr/bch/3.0.0/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.0-bc3.0.0.jar pi 1 100*

*18/11/16 22:07:56 INFO mapreduce.Job: Running job: job\_1542376844378\_0001*

*java.io.IOException: org.apache.hadoop.security.authorize.AuthorizationException: User hdfs (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.mapreduce.v2.api.MRClientProtocolPB: denied by configured ACL*

但是通过yarn用户提交程序时抛出异常如下：

*$ hadoop jar /usr/bch/3.0.0/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.0-bc3.0.0.jar pi 1 100*

*Number of Maps = 1*

*Samples per Map = 100*

*org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.AuthorizationException): User yarn (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.hdfs.protocol.ClientProtocol: denied by configured ACL*

由于提交程序需要访问hdfs，因此抛出异常，修改参数如下：

*<property>*

*<name>security.client.protocol.acl</name>*

*<value>\*</value>*

*</property>*

重启后则提交程序成功

*[yarn@cmhhost3 ~]$ hadoop jar /usr/bch/3.0.0/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.0-bc3.0.0.jar pi 1 100*

*18/11/16 22:23:24 INFO mapreduce.Job: Running job: job\_1542378080470\_0001*

*18/11/16 22:23:36 INFO mapreduce.Job: Job job\_1542378080470\_0001 running in uber mode : false*

*18/11/16 22:23:36 INFO mapreduce.Job: map 0% reduce 0%*

*18/11/16 22:23:50 INFO mapreduce.Job: map 100% reduce 0%*

*Job Finished in 42.242 seconds*

*Estimated value of Pi is 3.20000000000000000000*

## **1.4 Hadoop Service中支持Protocol ACL**

为了支持Service Level Authorization，需要在服务中定义服务及其定义的Protocol，以NameNode为例，其开启SLA后从hadoop-policy.xml中加载ACL信息：

*if (serviceAuthEnabled =*

*conf.getBoolean(CommonConfigurationKeys.HADOOP\_SECURITY\_AUTHORIZATION, false)) {*

*clientRpcServer.refreshServiceAcl(conf, new HDFSPolicyProvider());*

*if (serviceRpcServer != null) {*

*serviceRpcServer.refreshServiceAcl(conf, new HDFSPolicyProvider());*

*}*

*if (lifelineRpcServer != null) {*

*lifelineRpcServer.refreshServiceAcl(conf, new HDFSPolicyProvider());*

*}*

*}*

在Server.refreshServiceAcl中从配置中读取acl配置

*Service[] services = provider.getServices();*

*if (services != null) {*

*for (Service service : services) {*

*AccessControlList acl =*

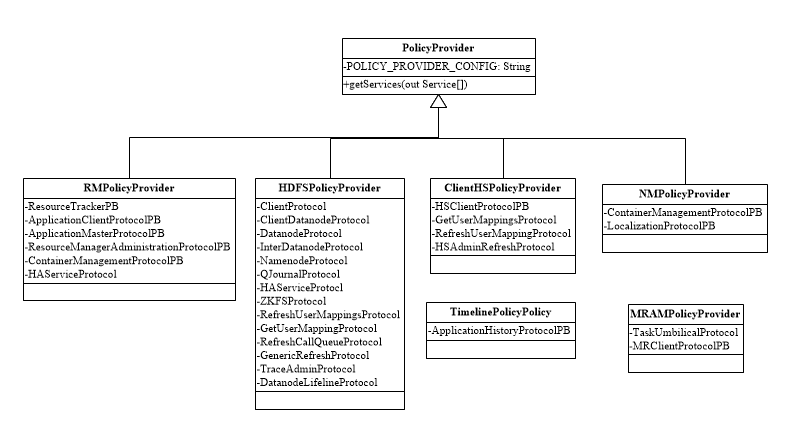
*new AccessControlList(*

*conf.get(service.getServiceKey(), //seviceKey为该service对应的acl配置参数*

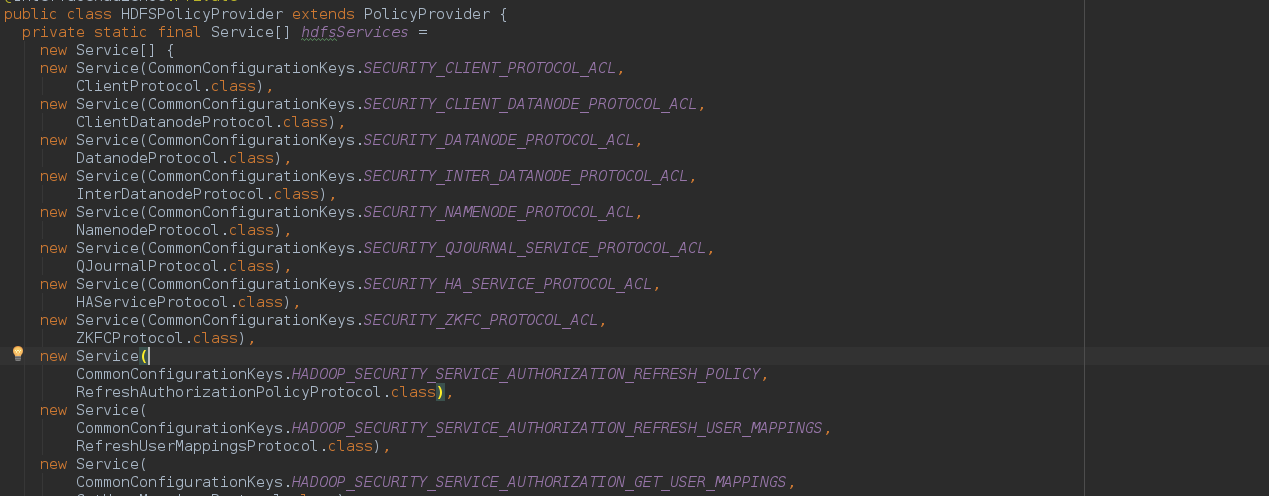
*defaultAcl)*

*);....}}*

PolicyProvider定义了Server(Protocol)与ACL参数的对应关系，PolicyProvider的定义如下：



NameNode中通过HDFSPolicyProvider定义了Server（Protocol）与acl参数的对应关系，PolicyProvider的定义如下：



PolicyProvider中的Service的类定义如下：

*public class Service {*

*private String key;*

*private Class<?> protocol;*

*}*

# 资源授权控制

在授权控制中，将Hadoop中的文件、队列、应用等作为授权控制的对象，资源授权控制分为两部分：

1. Hadoop分布式文件系统中的数据包括文件和文件夹，对数据的访问需要通过用户及用户组组合的身份凭证来进行数据访问的权限控制
2. YARN不负责对数据的访问控制，但是提供了集群资源的访问控制，如CPU、内存、磁盘及网络IO。YARN中通过资源队列来实现job的资源使用限制，为了实现这部分资源的精确控制，Hadoop在Job Queue中设置ACL列表机制。

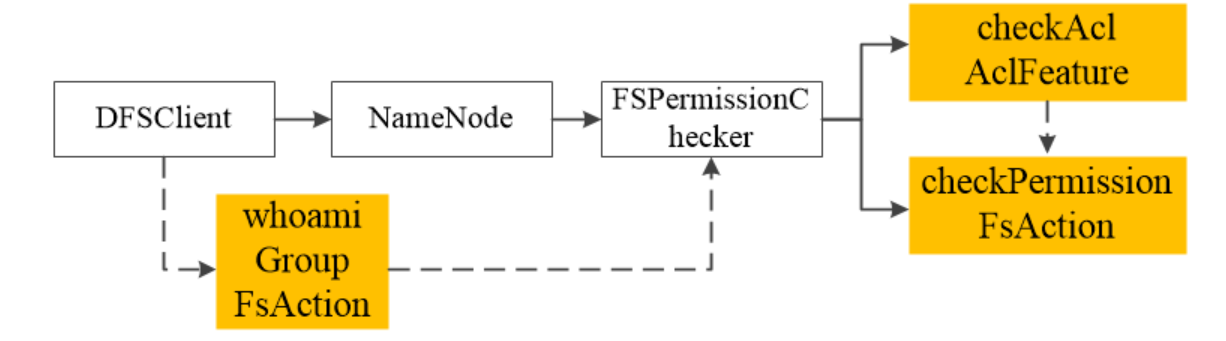
下面介绍这两种资源的控制逻辑。

## **2.1 HDFS权限控制**

HDFS数据的权限控制模型与POSIX文件系统的权限控制模型相同，每个文件和目录都有所有者和用户组。每个客户端访问HDFS的过程，身份凭证由用户名和组列表两部分组成，进行身份验证时首先验证用户名，如果用户名验证不通过则验证用户组。如果用户名和用户组都验证失败则身份验证失败。

但是Posix权限模型的控制仅能对数据所有者及所有组等进行控制比较粗粒度，HDFS还提供了对ACL(访问控制列表)的支持来为特定的用户或者用户组提供更加细粒度的文件权限。ACL是不同于用户和组的组织层次的权限控制方式，ACL可以为特定的用户和用户组设置不同的权限，而不是文件拥有者和文件所属组，默认情况下，HDFS禁用ACL。

客户端进行文件和目录的操作，要进行如下步骤：



1. 确认客户端所使用的用户，其通过宿主操作系统给出，类似Unix系统

* 用户名等于`whoami`
* 组类表等于`bash -c groups`

还有其他方式来确定用户身份，如Kerberos和LDAP等，客户端将用户和用户组封装成UGI后传递给NameNode

1. 客户端对文件和目录的操作都要传递完整的路径名给namenode，每一个操作都会此路径做权限检查
2. NameNode 进行文件目录权限的检查，FSpermissionCheck其操作如下：

*private boolean hasPermission(INodeAttributes inode, FsAction access) {*

*......*

*final FsPermission mode = inode.getFsPermission();*

*final AclFeature aclFeature = inode.getAclFeature();*

*if (aclFeature != null) { //ACL鉴权*

*// It's possible that the inode has a default ACL but no access ACL.*

*int firstEntry = aclFeature.getEntryAt(0);*

*if (AclEntryStatusFormat.getScope(firstEntry) == AclEntryScope.ACCESS) {*

*return hasAclPermission(inode, access, mode, aclFeature);*

*}*

*}*

*// FsPermission 鉴权，POSIX -RWX，与FsPermmission进行check*

*final FsAction checkAction;*

*if (getUser().equals(inode.getUserName())) { //user class*

*checkAction = mode.getUserAction();*

*} else if (isMemberOfGroup(inode.getGroupName())) { //group class*

*checkAction = mode.getGroupAction();*

*} else { //other class*

*checkAction = mode.getOtherAction();*

*}*

*return checkAction.implies(access);*

*}*

1. **HDFS Posix**

HDFS POSIX系统的文件和目录的权限模型，与UNIX文件权限类似使用位权限控制，其采用了Unix表示和显示模式的习惯格式如下所示：

*-rwxr-xr-x-. 1 fys fys 27797 Feb 26 2018 tmp*

权限类型<所有者权限,用户组权限,其他用户权限> |所有者|用户组|其他用户权限|文件大小|修改日期|文件名

* 权限三元组分为读(r)、写(w)和执行(x)，rwx代表文件所有者可以进行读写及执行的操作
* 文件所有者，创建文件的用户；文件所属组，属于哪个用户组；其他用户

HDFS支持Posix文件权限check，需要配置以下参数：

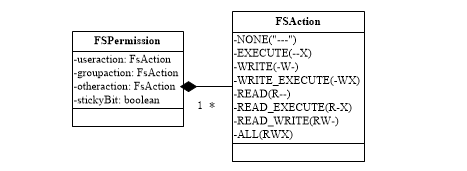
*<property>*

*<name>dfs.permissions.enabled</name>*

*<value>true</value>*

*</property>*

根据客户端传过来的FsAction进行Inode中保存的FSPermission信息进行check，其类图如下所示：



在INode中保存的permission信息（FSImage中持久化）格式如下：

*<inode>*

*<id>16385</id>*

*<type>DIRECTORY</type>*

*<name></name>*

*<mtime>1494453394091</mtime>*

*<permission>hdfs:hdfs:rwxr-xr-x</permission> //load到内存中，其对象为FSPermission*

*<nsquota>9223372036854775807</nsquota>*

*<dsquota>-1</dsquota>*

*</inode>*

1. HDFS ACL

在HDFS Posix权限验证中，单个文件只能从属于一个用户或者用户组，如果使该文件从属于多个组，该中权限验证机制就无法实现。在文件使用的过程中可能会遇到以下需求：

* 该文件对另外的组开发
* 组内某个人对该文件不可修改，ACL可以实现该文件对特定人或者特定组的一个权限管控，而不仅是文件的所属用户或者所属组

HDFS ACL与Linux中的ACL机制基本相同，都是用于为文件系统提供更精细化的权限控制。ACL权限控制只是对普通的权限控制的一个补充，其基础还是构建在位权控制上，因此尽量少使用ACL权限增加额外的控制。配置以下参数来开启ACL权限功能：

*<property>*

*<name>dfs.namenode.acls.enabled</name>*

*<value>true</value>*

*</property>*

在HDFS 权限控制中，访问控制列表由一组ACL组成，每个ACL项命名了特定的用户或者组，并为其授予或拒绝读写及执行权限。

下面是使用示例：文件/user/hdfs/acl.txt所属者为hdfs，通过设置acltest用户acl的让其有读写权限，执行如下：

*$ hdfs dfs -getfacl /user/hdfs/acl.txt*

*# file: /user/hdfs/acl.txt*

*# owner: hdfs*

*# group: hdfs*

*user::rw-*

*group::r--*

*other::---*

*$ hdfs dfs -cat /user/hdfs/acl.txt*

*cat: Permission denied: user=acltest, access=READ, inode="/user/hdfs/acl.txt":hdfs:hdfs:-rw-r-----*

*[hdfs@cmhhost1 ~]$ hdfs dfs -setfacl -m user:acltest:rwx /user/hdfs/acl.txt*

*[hdfs@cmhhost1 ~]$ hdfs dfs -getfacl /user/hdfs/acl.txt*

*# file: /user/hdfs/acl.txt*

*# owner: hdfs*

*# group: hdfs*

*user::rw-*

*user:acltest:rwx*

*group::r--*

*mask::rwx*

*other::---*

*[acltest@cmhhost1 ~]$ hdfs dfs -cat /user/hdfs/acl.txt*

*hello*

*acl*

ACL项由类型,可选名称和权限字符串组成，它们之间使用冒号”:”隔开，上例中文件所有者具有读写权限、文件组具有读权限，但是其他用户没有权限。上例中文件的ACL项：

*user::rw-*

*user:acltest:rwx*

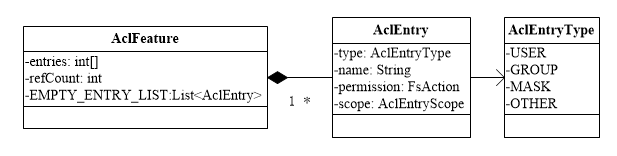
*group::r--*

*mask::rwx*

*other::---*

默认ACL必须包含所有最小的ACL项，包括文件拥有者项、文件所属组合其他用户想，如果用户没有在默认的ACL中配置上述三项任何一个，则从位权控制中自动导入。

ACL项在FsImage中的INode#AclFeature中保存，其相关类图如下：



在FsImage文件中保存数据如下：

*<inode>*

*<id>16522</id>*

*<type>FILE</type>*

*<name>acl.txt</name>*

*<acls>*

*<acl>user:acltest:rwx</acl>*

*<acl>group::r--</acl>*

*</acls>*

*.....*

*</inode>*

FSpermissionCheck#hsAclPermission的执行如下：

*private boolean hasAclPermission(INodeAttributes inode,*

*FsAction access, FsPermission mode, AclFeature aclFeature) {*

*boolean foundMatch = false;*

*// 如果User是文件所有者，则直接调用FsPermission*

*if (getUser().equals(inode.getUserName())) {*

*if (mode.getUserAction().implies(access)) {*

*return true;*

*}*

*foundMatch = true;*

*}*

*// 使用ACL进行验证.*

*if (!foundMatch) {*

*for (int pos = 0, entry; pos < aclFeature.getEntriesSize(); pos++) {*

*entry = aclFeature.getEntryAt(pos);*

*if (AclEntryStatusFormat.getScope(entry) == AclEntryScope.DEFAULT) {*

*break; }*

*AclEntryType type = AclEntryStatusFormat.getType(entry);*

*String name = AclEntryStatusFormat.getName(entry);*

*//如果ACL Type是USER，则验证USER*

*if (type == AclEntryType.USER) {*

*if (getUser().equals(name)) {*

*FsAction masked = AclEntryStatusFormat.getPermission(entry).and(*

*mode.getGroupAction());*

*if (masked.implies(access)) {*

*return true;}*

*foundMatch = true;*

*break;*

*}*

*} else if (type == AclEntryType.GROUP) {*

*//ACL Type为GROUPS*

*String group = name == null ? inode.getGroupName() : name;*

*if (isMemberOfGroup(group)) {*

*FsAction masked = AclEntryStatusFormat.getPermission(entry).and(*

*mode.getGroupAction());*

*if (masked.implies(access)) {*

*return true;*

*}*

*foundMatch = true;*

*}}}}*

*return !foundMatch && mode.getOtherAction().implies(access);*

*}*

## **2.2 YARN 资源授权**

YARN通过Job队列来实现资源的分配，其授权包括配置用户向队列提交程序的权限及用户管理队列的权限。YARN授权机制是通过访问控制列表（ACL）来实现，按照授权实体，将ACL列表分为：

1. 队列访问控制列表，为了方便管理集群中的用户，将用户/用户组分成若干队列，并可指定每个用户/用户组所属的队列。每个队列包含两种权限：提交应用程序权限和管理应用程序权限
2. 应用程序访问控制列表，在客户端为应用设置ApplicationAccessType(目前只有View和Modify两种类型)对应的用户列表。这些信息传递给RM后，由其维护和使用。对于MR，用户设置参数mapreduce.job.acl-view-job和mapreduce.job.acl-modify-job为每个队列单独设置查看和修改权限
3. 服务访问控制列表，服务访问控制是Hadoop提供的最原始的授权机制，在SLA中已经介绍

下面分析基于FairScheduler对不同队列实现不同用户提交权限。启用YARN 队列访问控制权限，设置参数如下

*<property>*

*<name>yarn.acl.enable</name>*

*<value>true</value>*

*</property>*

在fair-scheduler.xml文件中配置队列及ACL，示例如下：

*<?xml version="1.0" encoding="utf-8"?>*

*<allocations>*

*<defaultQueueSchedulingPolicy>drf</defaultQueueSchedulingPolicy>*

*<queue name="root">*

*<queue name="ambari-qa">*

*<aclAdministerApps>ambari-qa</aclAdministerApps>*

*<aclSubmitApps>ambari-qa</aclSubmitApps>*

*</queue>*

*<queue name="acltest">*

*<aclAdministerApps>hdfs</aclAdministerApps> //只运行hdfs用户管理程序*

*<aclSubmitApps>yarn</aclSubmitApps> //允许yarn用户提交程序*

*</queue>*

*</queue>*

*</allocations>*

向root.acltest队列提交程序，执行如下：

*[hdfs@cmhhost1 ~]$ hadoop jar /usr/bch/3.0.0/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.0-bc3.0.0.jar pi -Dmapreduce.job.queuename=root.acltest 1 1000*

*Number of Maps = 1*

*18/11/18 10:17:06 INFO mapreduce.Job: Running job: job\_1542378080470\_0002*

*java.io.IOException: org.apache.hadoop.security.authorize.AuthorizationException: User hdfs (auth:SIMPLE) is not authorized for protocol interface org.apache.hadoop.mapreduce.v2.api.MRClientProtocolPB: denied by configured ACL*

使用hdfs提交程序触发AuthorizationException。使用yarn用户提交程序，执行正常：

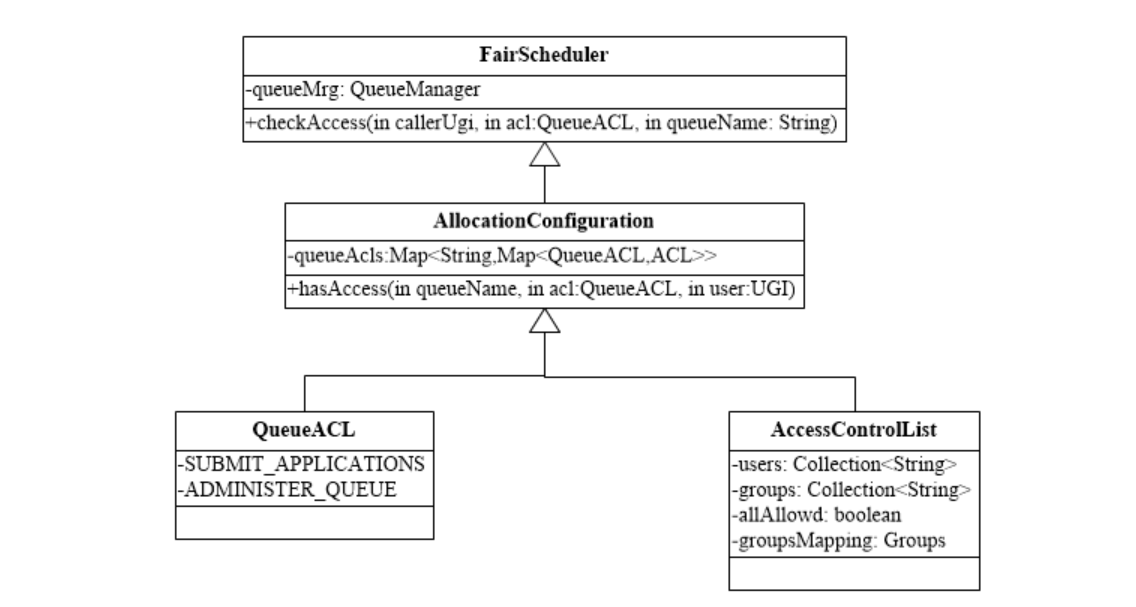
*[yarn@cmhhost1 ~]$ hadoop jar /usr/bch/3.0.0/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.0-bc3.0.0.jar pi -Dmapreduce.job.queuename=root.acltest 1 1000*

*Starting Job*

*Job Finished in 34.721 seconds*

*Estimated value of Pi is 3.14800000000000000000*

YARN队列访问控制列表（ACL）保存在队列信息中，相关类图如下所示：



将程序提交到队列时，FairScheduler进行acl check，其执行如下：

*@Override*

*public synchronized boolean checkAccess(UserGroupInformation callerUGI,*

*QueueACL acl, String queueName) {*

*FSQueue queue = getQueueManager().getQueue(queueName);*

*if (queue == null) { return false;}*

*return queue.hasAccess(acl, callerUGI);*

*}*

其最终调用AllocationConfiguration#check，其从fair-scheduler.xml中读取ACL信息，并进行验证：

*public boolean hasAccess(String queueName, QueueACL acl,*

*UserGroupInformation user) {*

*int lastPeriodIndex = queueName.length();*

*while (lastPeriodIndex != -1) {*

*String queue = queueName.substring(0, lastPeriodIndex);*

*if (getQueueAcl(queue, acl).isUserAllowed(user)) {*

*return true;*

*}*

*lastPeriodIndex = queueName.lastIndexOf('.', lastPeriodIndex - 1);*

*}*

*return false;*

*}*

http://hadoop.apache.org/docs/r1.0.4/cn/hdfs\_permissions\_guide.html

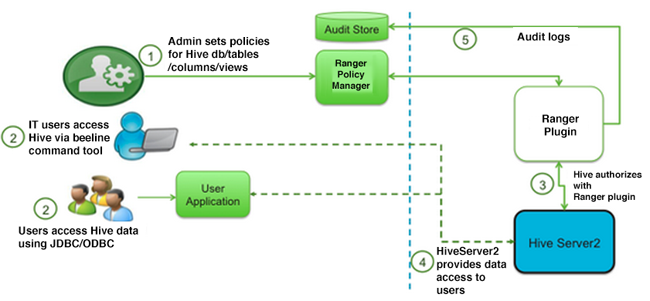
https://blog.csdn.net/skywalker\_only/article/details/40709447

https://blog.csdn.net/u010003835/article/details/71172873

# ThirtParty授权机制

Hadoop中资源的权限控制有相应模块进行管理，其控制策略不同而且分散在不同模块中很难去维护，这就需要集中式安全管理框架，目前主流的第三方安全管理框架有Apache Ranger和Sentry。集中式安全管理框架针对各种资源提供了统一的访问权限模型和管理界面，使集群管理员可以轻松的通过配置策略来控制各资源的访问权限。这里介绍Ranger的原理及使用。

Apache Ranger鉴权的本质是通过读取安全组件时生成的配置文件以及组件自带的jar包，通过hook的方式调用各个组件服务达到权限管理。在安装服务插件的过程中，主要指向三个步骤：将插件自带的conf更新到系统安装的服务conf下；将插件自带的lib更新到系统安装的服务lib下；将install.properties生成的xml文件，更新到系统安装的服务conf下。其Ranger权限管理流程图分析（以hive为例）：



如上图，鉴权过程分为5个步骤：

1. 集群管理者为Hive数据库设置安全策略
2. 集群访问者通过beeline或者JDBC访问Hive中的数据
3. 访问请求通过Ranger plugin进行安全验证，该插件具体的执行流程在下面会进行具体分析
4. 验证通过后，HiveServer2向用户访问请求数据
5. Range plugin根据用户和请求生成审计日志

服务的权限管理的执行逻辑是通用的，将Agent Plugin的鉴权逻辑嵌入到决策流程中是Apache Ranger的实现核心。

## 3.1权限模型

权限模型无非是定义了”用户-资源-权限”这三者的关系，Ranger基于Policy来抽象这种关系，进而扩展本身的权限模型。为了简化模型，使用下面的表达式来描述：

*Policy = Service + List<Resource> + AllowACL+ DenyACL*

*AllowACL = List<AccessItem> allow + List<AccessItem> allowException*

*DenyACL = List<AccessItem> deny + List<AccessItem> denyException*

*AccessItem = List<User/Group> +List<AccessType>*

从”用户-资源-权限”的角度进行详解:

1. 用户，由User或Group来表达，User代表访问资源的用户，Group代表用户所属的用户组
2. 资源，<Service,Resource>二元组来表达，一条Policy唯一对应一个Service，但可以对应多个Resource
3. 权限，由<AllowACL,DenyACL>二元组来表达，两者都包含两组AccessItem，AccessItem则描述一组用户与一组访问之间的关系，在AllowACL中表示允许执行，而DenyACL中表示拒绝执行

下面列出几种常见系统的模型实体枚举值：

|  |  |  |
| --- | --- | --- |
| Service | Resource | Access Type |
| HDFS | Path | Read/Write/Execute |
| YARN | Queue | Submit/Admin |
| HBase | Table,Column Family,Column | Read/Write/Create/Admin |
| Hive | Database,Table,Column | Select/Update/Created/Drop  /Alter/Index/Lock |

以AllowACL为例，假定要将资源授权给一个用户组G1，但是用户组里的某个用户U1除外，只需要增加一条包含G1的AccessItem到AllowACL\_allow，同时增加一条包含U1的AccessItem到AllowACL\_allowException即可。

一条Policy <= {alow,allowException,deny,denyException}四组AccessItem，作用的优先级由高到低依次：denyException > deny > allowException>allow，访问决策树流程图如下：



如果没有Policy能决策访问，Ranger可以选择将决策下放给系统自身的访问控制层，比如HDFS的ACL。

## 3.2 Agent Plugin的嵌入

将Agent Plugin的鉴权逻辑嵌入到决策流程有两种方法：

1. **实现可扩展的接口**

多数的系统在实现时都有考虑功能扩展性的问题，一般会为核心模块暴露出可扩展的接口，访问控制模块也不例外。Ranger通过实现访问控制接口，将自己的逻辑嵌入到各个系统。下表列出了Ranger插件对几个常见系统的扩展接口：

|  |  |
| --- | --- |
| Service | Extensible Interface |
| YARN | org.apache.hadoop.yarn.security.YarnAuthorizionProvider |
| HBase | org.apache.hadoop.hbase.coprocessor.RegionObserver |
| Hive | org.apache.hadoop.hive.ql.security.authorization.plugin.HiveAuthorizer |
| Kafka | kafka.security.auth.Authorizer |
| Storm | backtype.storm.security.auth.IAuthorizer |
| Solr | org.apache.solr.security.AuthorizationPlugin |

1. **代码注入**

不排除少数系统没有将暴露访问控制扩展点，这种情况下，Ranger依赖Java代码注入机制（java.lang.instrument）来实现逻辑嵌入。

以HDFS插件为例，Ranger使用ClassFileTransformer，直接修改HDFS访问控制类的FSPermissionChecker的clsss File，将checkPermission方法替换成Ranger的自定义实现。其实现框架如下所示：

代码Inject



HDFS文件认证流程

参考链接：

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ServiceLevelAuth.html