# **[Kerberos协议](http://www.itnose.net/detail/6505517.html)**

**Kerberos协议**

**定义**

一种计算机网络授权协议，用来在非安全网路中，对个人通信以安全的手段进行身份认证。这个词又指麻省理工学院为这个协议开发的一套计算机软件。软件设计上采用客户端/服务器结构，并且能够进行相互认证，即客户端和服务器端均可对对方进行身份认证。可以用于防止窃听、防止replay攻击、保护数据完整性等场合，是一种应用对称密钥体制进行密钥管理的系统。Kerberos的扩展产品也使用公开密钥加密方法进行认证。

**基本概念**

Princal(安全个体)：被认证的个体，有一个名字和口令

KDC(key distribution center ) : 是一个网络服务，提供ticket 和临时会话密钥

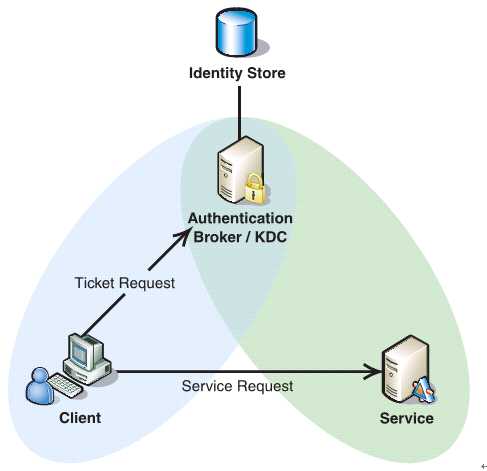
Ticket：一个记录，客户用它来向服务器证明自己的身份，包括客户标识、会话密钥、时间戳。

AS (Authentication Server)： 认证服务器

TSG(Ticket Granting Server)： 许可证服务器

**使用场景**

如下图所示，Client与KDC， KDC与Service 在协议工作前已经有了各自的共享密钥，并且由于协议中的消息无法穿透防火墙，这些条件就限制了Kerberos协议往往用于一个组织的内部， 使其应用场景不同于X.509 PKI。



**过程**  
  
Kerberos协议分为两个部分：

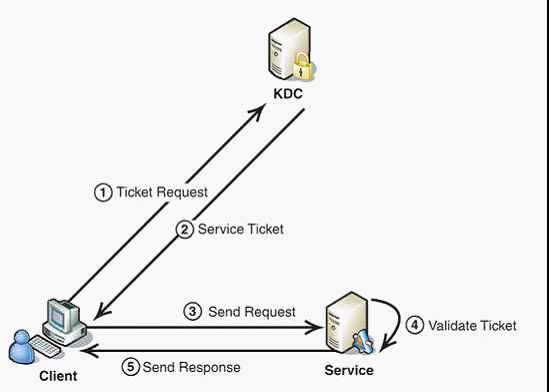
1 . Client向KDC发送自己的身份信息，KDC从Ticket Granting Service得到TGT(ticket-granting ticket)， 并用协议开始前Client与KDC之间的密钥将TGT加密回复给Client。

此时只有真正的Client才能利用它与KDC之间的密钥将加密后的TGT解密，从而获得TGT。

（此过程避免了Client直接向KDC发送密码，以求通过验证的不安全方式）

2. Client利用之前获得的TGT向KDC请求其他Service的Ticket，从而通过其他Service的身份鉴别。

 Kerberos协议的重点在于第二部分，简介如下：



1．    Client将之前获得TGT和要请求的服务信息(服务名等)发送给KDC，KDC中的Ticket Granting Service将为Client和Service之间生成一个Session Key用于Service对Client的身份鉴别。然后KDC将这个Session Key和用户名，用户地址（IP），服务名，有效期, 时间戳一起包装成一个Ticket(这些信息最终用于Service对Client的身份鉴别)发送给Service， 不过Kerberos协议并没有直接将Ticket发送给Service，而是通过Client转发给Service.所以有了第二步。

2．    此时KDC将刚才的Ticket转发给Client。由于这个Ticket是要给Service的，不能让Client看到，所以KDC用协议开始前KDC与Service之间的密钥将Ticket加密后再发送给Client。同时为了让Client和Service之间共享那个秘密(KDC在第一步为它们创建的Session Key)， KDC用Client与它之间的密钥将Session Key加密随加密的Ticket一起返回给Client。

3．    为了完成Ticket的传递，Client将刚才收到的Ticket转发到Service. 由于Client不知道KDC与Service之间的密钥，所以它无法算改Ticket中的信息。同时Client将收到的Session Key解密出来，然后将自己的用户名，用户地址（IP）打包成Authenticator用Session Key加密也发送给Service。

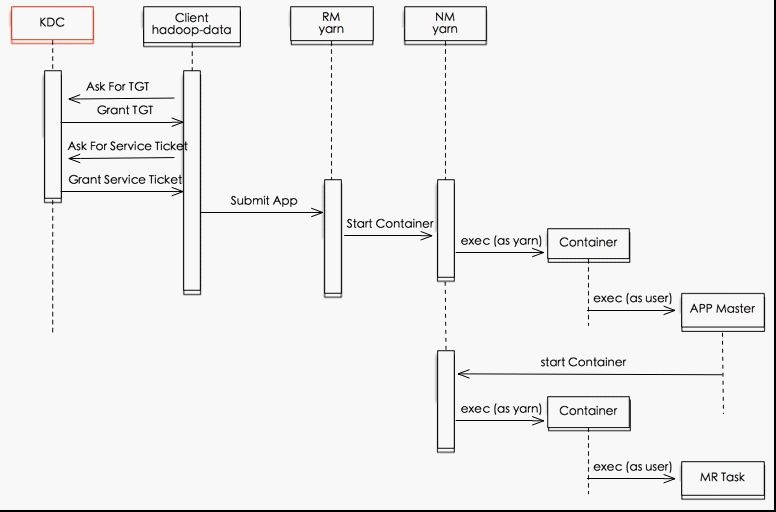
4．    Service 收到Ticket后利用它与KDC之间的密钥将Ticket中的信息解密出来，从而获得Session Key和用户名，用户地址（IP），服务名，有效期。然后再用Session Key将Authenticator解密从而获得用户名，用户地址（IP）将其与之前Ticket中解密出来的用户名，用户地址（IP）做比较从而验证Client的身份。

5．    如果Service有返回结果，将其返回给Client。

**Hadoop安全认证案例**

Hadoop 的安全认证是基于 [Kerberos](http://web.mit.edu/kerberos/" \t "http://www.itnose.net/detail/_blank) 实现的。 Kerberos 是一个网络身份验证协议，用户只需输入身份验证信息，验证通过获取Ticket即可访问多个接入 Kerberos 的服务， 机器的单点登录也可以基于此协议完成的。 Hadoop 本身并不创建用户账号，而是使用 Kerberos 协议来进行用户身份验证，从Kerberos凭证中的用户信息获取用户账号， 这样一来跟实际用户运行的账号也无关。

这里我们从 YARN 上的 MR 任务提交过程简单说明一下:



1 用户执行任务前，先通过KDC认证自己，获取TGT(Ticket Granting Ticket)。KDC是 Kerberos 认证的中心服务，存储用户和服务的认证信息。

2 用户通过 TGT 向 KDC 请求访问服务的Ticket， KDC 生成 session key 后一并发给客户端。

3 客户端通过 service ticket 向服务认证自己，完成身份认证。

4 完成身份认证后客户端向服务请求若干token供后续任务执行认证使用(比如 HDFS NameNode Delegation Token, YARN ResourceManager Delegation Token)

5 客户端连同获取到的 token 一并提交任务，后续任务执行使用 token 进行来自服务的认证

从上可以看出，出于性能的考虑，Hadoop 安全认证体系中仅在用户跟服务通信以及各个服务之间通信适用 Kerberos 认证，在用户认证后任务执行，访问服务，读取/写入数据等均采用特定服务(NameNode， Resource Manager)发起访问token，让需求方凭借 token 访问相应服务和数据。这里 token 的传递，认证以及更新不做深入讨论。

**客户端针对安全认证的相应调整**

集群开启安全认证之后， 依赖集群的客户端(脚本， 服务)都需要做相应修改，不过改动基本无异。大部分服务都已包括对 Kerberos 认证的相应处理， 基本不需要修改。

这里首先得说明一下开启安全认证后的认证方式:

· 使用密码认证  
使用用户密码通过kinit认证， 获取到的TGT存在本地凭证缓存当中， 供后续访问服务认证使用。一般在交互式访问中使用。

· 使用 [keytab](http://kb.iu.edu/data/aumh.html" \t "http://www.itnose.net/detail/_blank) 认证  
用户通过导出的keytab 可以免密码进行用户认证， 后续步骤一致。一般在应用程序中配置使用。

Kerberos 凭证(ticket) 有两个属性， ticket\_lifetime 和 renew\_lifetime。其中 ticket\_lifetime 表明凭证生效的时限，一般为24小时。在凭证失效前部分凭证可以延期失效时间(即Renewable)， renew\_lifetime 表明凭证最长可以被延期的时限，一般为一个礼拜。当凭证过期之后，对安全认证的服务的后续访问则会失败。这里第一个问题就是如何处理凭证过期。

凭证过期处理策略

在最早的 [Security features for Hadoop](https://issues.apache.org/jira/browse/HADOOP-4487" \t "http://www.itnose.net/detail/_blank) 设计中提出这样的假设:

A Hadoop job will run no longer than 7 days (configurable) on a MapReduce cluster or accessing HDFS from the job will fail.

对于一般的任务， 24小时甚至延迟到一周的凭证时限是足够充分的。所以大部分时间我们只需要在执行操作之前使用 kinit 认证一遍，再起后台任务进行周期性凭证更新即可。

while true ; do kinit -R; sleep $((3600 \* 6)) ; done &

不过对于需要常驻的访问Hadoop集群的服务来说，上述假设就不成立了。这时候我们可以

扩大 ticket\_lifetime 和 renew\_lifetime 时限  
扩大凭证存活时限可以解决此问题，但由于Kerberos跟我们线上用户登陆认证绑定，会带来安全隐患，故不方便修改。

定期重新进行kinit 认证更新凭证  
不仅仅是定期延长认证时间，可以直接定期重新认证以延长凭证有限期限。一般我们需要导出 keytab 来进行定期认证的操作。

Hadoop 将 Kerberos 认证部分进行了一定的封装，实际上并不需要那么复杂， 这里重点可以看看UserGroupInformation 这个类。

UserGroupInformation

UserGroupInformation 这个类在 [JAAS](http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html" \l "Core" \t "http://www.itnose.net/detail/_blank) 框架上封装了 Hadoop 的用户信息， 更确切地说是对 Subject 做了一层封装。

 UserGroupInformation(Subject subject) {

    this.subject = subject;

    this.user = subject.getPrincipals(User.class).iterator().next();

    this.isKeytab = !subject.getPrivateCredentials(KerberosKey.class).isEmpty();

    this.isKrbTkt = !subject.getPrivateCredentials(KerberosTicket.class).isEmpty();

  }

JAAS 是 Java 认证和授权服务（Java Authentication and Authorization Service）的缩写， 主要包含以下几个实体:

· Subject  
Subject 是一个不可继承的实体类，它标志一个请求的来源， 包含相关的凭证标识(Principal) 和 公开和私有的凭据(Credential)。

· Principal  
凭证标识，认证成功后，一个 Subject 可以被关联多个Principal。

· Credential  
凭据，有公有凭据以及私有凭据。

JAAS的认证过程如下:

1 An application instantiates a LoginContext.

2 The LoginContext consults a Configuration to load all of the LoginModules configured for that application.

3 The application invokes the LoginContext's login method.

4 The login method invokes all of the loaded LoginModules. Each LoginModule attempts to authenticate the subject. Upon success， LoginModules associate relevant Principals and credentials with a Subject object that represents the subject being authenticated.

5 The LoginContext returns the authentication status to the application.

6 If authentication succeeded， the application retrieves the Subject from the LoginContext.

需要认证的代码片段可以包装在 [doPrivileged](http://docs.oracle.com/javase/6/docs/technotes/guides/security/doprivileged.html" \t "http://www.itnose.net/detail/_blank) 当中， 可以直接使用 Subject.doAs 方法，支持嵌套。

在安全模式下，UGI 支持不同LoginContext 配置， 均是通过 HadoopConfiguration 类动态产生:

· hadoop-user-kerberos  
使用kerberos缓存凭证登陆的配置， useTicketCache 置为 true.

· hadoop-keytab-kerberos  
使用keytab登陆的配置， useKeyTab 置为 true.

UGI 当中有多处认证， getLoginUser 方法使用 hadoop-user-kerberos 配置认证:

1 通过配置生成 LoginContext

2 调用 LoginContext.login 方法完成登陆， 通过 ticket cache 中凭证完成登陆

3 [判断是否需要其他用户身份(proxy user)执行](https://issues.apache.org/jira/browse/HADOOP-8561" \t "http://www.itnose.net/detail/_blank)

4 将 HADOOP\_TOKEN\_FILE\_LOCATION 中的 token 加入 Credentials 集合当中

5 另起一个线程做周期性的凭证更新 spawnAutoRenewalThreadForUserCreds

步骤5可以看出当我们存在凭证后并不需要主动做周期性地凭证更新。

而 loginUserFromKeytab 方法使用 hadoop-kerberos 配置认证:

1 通过配置生成 LoginContext

2 调用 LoginContext.login 方法完成登陆， 使用keytab完成登陆

loginUserFromKeytab 没有对凭证做周期的更新， 那怎么保证凭证不会过期呢?

1 在访问集群执行相关操作前， 可以调用 checkTGTAndReloginFromKeytab 来尝试更新凭证(实际上是重新登陆了)

2 在凭证过期时，创建 IPC 失败会触发调用 reloginFromKeytab 来重新登陆

Client.java

   private synchronized void handleSaslConnectionFailure(

        final int currRetries， final int maxRetries， final Exception ex，

        final Random rand， final UserGroupInformation ugi) throws IOException，

        InterruptedException {

      ugi.doAs(new PrivilegedExceptionAction<Object>() {

        @Override

        public Object run() throws IOException， InterruptedException {

          final short MAX\_BACKOFF = 5000;

          closeConnection();

          disposeSasl();

          if (shouldAuthenticateOverKrb()) {

            if (currRetries < maxRetries) {

              if(LOG.isDebugEnabled()) {

                LOG.debug("Exception encountered while connecting to "

                    + "the server : " + ex);

              }

              // try re-login

              if (UserGroupInformation.isLoginKeytabBased()) {

                UserGroupInformation.getLoginUser().reloginFromKeytab();

              } else {

                UserGroupInformation.getLoginUser().reloginFromTicketCache();

              }

可见如果是使用 keytab 认证的话，认证是长期有效的。

从上述代码中可以看到，不论是否是keytab认证，创建IPC失败均会尝试重新登陆。

基于keytab 的Kerberos认证方式

为了让用户免于记忆密码，我们可以考虑导出并交付keytab给相关用户(前提是用户数量可控， 比如是以虚拟用户为单位)。  
这样，用户的Hadoop任务认证方式可以有:

· 直接使用 keytab kinit 之后访问

· 或者调用 loginUserFromKeytab 完成登录，然后将代码片段包裹在 UGI 的 doAs 方法当中执行