# Apache Kerberos概述

## Kerberos技术概况

Kerberos协议是一种计算机网络授权协议，用于在非安全的网络环境下对个人通信进行加密认证，Kerberos协议认证过程的实现不依赖于主机操作系统的认证，无需基于主机地址的信任，不要求网络上所有主机的物理安全，并且是以假定网络上传送的数据包可以被任意的读取、修改和插入数据为前提设计的。

Kerberos设计的目标如下：

1. 用户的密码不能在网络上传输；
2. 用户的密码决不能以任何形式存储在客户机上，使用后必须立刻销毁；
3. 用户的密码不应该以未加密的形式被存储；
4. 用户在一次工作会话期间只会被要求输入一次密码，因此用户能够在这次会话期间透明的访问所有服务而无需重新输入密码。
5. 认证信息的管理集中在认证服务器上，应用服务器上不能保留用户的身份验证信息，以此来实现以下的功能。

* 管理员可以通过修改认证服务器的设置来对指定用户的账户进行修改，而不用修改应用服务器。
* 当用户修改密码时，将在所有服务器上生效
* 没有需要保护的冗余信息
* 不仅用户需要向应用服务器证明他们的身份，应用服务器在返回时也需要向客户端证明他们的真实性，实现双向认证。
* 在完成身份的验证后，客户端与服务器必须能够建立一个加密的连接，因此Kerberos需要为生成和交换加密秘钥提供支持。

## Kerberos发展史及近况

在希腊神话中Kerberos是守护地狱之门的一条凶猛的三头神犬，而我们在本文中所要介绍的Kerberos认证协议是由美国麻省理工学院(MIT)首先提出并实现的，是该校雅典娜计划的一部分。这个定名是贴切的，因为**Kerberos认证是一个三路处理过程**，依赖称为密钥分发中心（KDC）的第三方服务来验证计算机相互的身份，并建立密钥以保证计算机间安全连接。本质上每台计算机分享KDC一个秘钥，而KDC有两个部件：一个Kerberos 认证服务器和一个票据授权服务器。如果KDC不知道被请求目标服务器，则会求助于另一个KDC来完成认证。它允许在网络上通讯的实体互相证明彼此的身份，并且能够阻止窃听和重放等攻击手段。不仅如此，它还能够提供对通讯数据保密性和完整性的保护。

Kerberos从提出到今天，共经历了五个版本的发展。其中第一版到第三版主要由该校内部使用。当发展到第四版的时候，已经取得了在MIT校外的广泛认同和应用。由于第四版的传播，人们逐渐发现了它的一些局限性和缺点，例如适用网络环境有限、加密过程存在冗余等等。MIT充分吸取了这些意见，对第四版进行了修改和扩充，形成了今天非常完善的第五版。第五版由John Kohl和Clifford Neuman设计，在1993年作为[RFC 1510](http://tools.ietf.org/html/rfc1510)颁布，在2005年由[RFC 4120](http://tools.ietf.org/html/rfc4120)取代，目的在于克服第四版的局限性和安全问题，用更细化和明确的解释说明了协议的一些细节和使用方法。

## Kerberos架构

Kerberos由Kerberos服务器KDC、客户机（Client）和应用服务器（Application Server）组成，其中KDC包含了Authorization Service和Ticket Granting Service两种服务。

1. KDC：Key Distribution Center，即密钥分发中心，就是通常所说的认证服务器，他是参与用户和服务认证的基本对象，由于他有密钥分配的功能并且可以作为服务接入，因此被称为密钥分发中心，简称KDC，KDC通常是一台单独的物理服务器，他可以从逻辑上分为3个部分：数据库（Database）、认证服务（Authorization Service）和票据授权服务（Ticket Granting Service）。
2. Database：数据库，用于存放用户和服务的记录，Kerberos的数据库中使用Principal来命名和引用一条记录，Kerberos数据库中的记录包含以下内容：

* 记录所关联的Principal
* 加密密钥和相关的KVNO
* 与Principal关联的票据的最长有效期
* 与Principal关联的票据的最长更新周期
* 描述票据的参数或标志
* 密码过期时间
* Principal的过期时间

1. Application Server：应用服务器，应用服务器上需要安装Kerberos客户端，在相关服务中开启对Kerberos协议的支持，并且对服务所使用的Principal进行配置。
2. Client：客户机，用户使用客户机来获得应用服务器提供的各项服务，客户机上需要安装Kerberos客户端，在使用时，用户需要先向KDC进行身份的认证，才能从应用服务器获得相应的服务。
3. AS：Authorization Service，认证服务，认证服务是KDC中用户回复客户端最初的认证请求的部分，如果用户没有认证过，必须输入密码，在回应认证请求时，AS会授予一个特殊的被称为票据授权的票据（Ticket Granting Ticket）。简称TGT。如果用户确实是他们所声称的身份，他们就可以使用TGT在无需再次输入密码的情况下获得其他服务的票据。
4. TGS：Ticket Granting Service，票据授权服务，票据授权服务是KDC中负责根据用户提交的有效TGT分配服务票据（Service Ticket）的组件，同时TGS保证向应用服务器请求资源的身份的真实性，TGS可以被视为一个应用服务器，提供服务票据的功能。
5. ST：Service Ticket，服务票据，由KDC的TGS发放，任何一个应用（Application）都需要一张有效的服务票据才能访问，如果能正确接收ST，说明Client和Server之间的信任关系已经被建立，ST通常为一张数字加密的证书。
6. TGT：Ticket Granting Ticket，票据授权票据，由KDC的AS发放，获得这样一张票据后，再申请其他应用的服务票据（ST）时，就不需要向KDC提交身份认证信息（Credential），TGT具有一定的有效期，到期后需要更新来续约。
7. Ticket：票据。票据是客户端提交该应用服务器用于证明其身份真实性的，票据由认证服务器颁发，并使用所需要的服务端密钥加密。由于服务端密钥只存在于认证服务器于应用服务器之间，获取到该票据的客户端也无法知道服务端密钥，因而也无法对票据进行修改。一张票据中包含了以下的信息：

* 请求用户的Principal（一般来说是用户名）
* 用户所请求的服务Principal
* 可以使用该票据的客户端IP地址（可选）
* 票据的生效日期与时间（使用时间戳格式）
* 票据的有效时间
* 会话密钥（Session Key）

每张票据都会有过期时间（通常为10小时），这是必要的，虽然认证服务器的管理员可以控制不再发布新的票据，但无法阻止用户使用已经发布的票据，因为认证服务器无法对已经发布的票据进行控制，所以过期时间的设置可以防止票据被滥用。

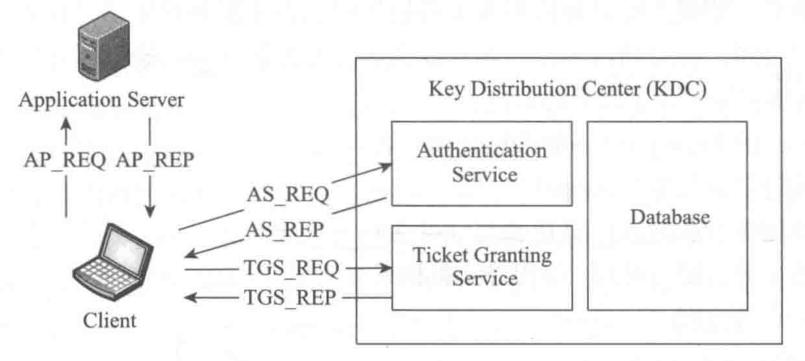
1. KVNO：Key Version Number，密钥版本号，当用户更改密钥或管理员更新应用服务器的密钥时，这种修改将会使版本号增加。

## 认证流程

下面将会对认证过程中被传递的数据包进行说明，以此来描述Kerberos认证的简要流程，需要强调的是，应用服务器在整个过程中不会直接与KDC进行通信，服务票据、TGS等数据全部由客户端发送至它们所需要的应用服务器。

Kerberos认证流程如图所示，客户端传输的数据包顺序是从上至下，其中数据包内容如下：

1. AS\_REQ：AS\_REQ是最初的用户认证请求，由kinit命令生成，这个信息指向KDC中的AS服务。
2. AS\_REP：AS\_REP是AS服务对先前请求的答复，它主要包含TGT（使用TGS密钥加密）和会话密钥SK\_TGS（使用请求认证的用户密钥加密）
3. TGS\_REQ：TGS\_REQ是客户端向TGS服务请求服务票据（ST）的请求包，这个数据包包含了之前的TGT数据以及一个由会话密钥加密过的客户端身份凭证。
4. TGS\_REP：TGS\_REP是TGS服务对之前请求的答复，其中包含了客户端所请求的服务票据（使用服务密钥加密）以及一个由TGS生成的会话密钥SK\_Service（使用之前AS服务生成的会话密钥加密）。
5. AP\_REQ：AP\_REQ是客户端发送给应用服务器用于使用服务的请求，其中包含了之前从TGS服务请求到的服务票据以及由客户端生成的身份凭证，但这次使用由TGS生成的会话密钥（SK\_Service）加密。
6. AP\_REP：AP\_REP由应用服务器发送给客户端，证明自己确实是用户所请求的服务端，这个数据包是可选的，只有在使用了相互认证机制时客户端才会请求这个数据包。



1. User向KDC中的AS请求身份验证，AS为user和TGS生成一个session key：SK\_TGS，并发送{ TGT, SK\_TGS } K\_USER；其中，{TGT, SK\_TGS}K\_USER表示使用user的密码加密的packet，包含了TGT和用户与TGS的session key；这个请求验证的过程实际上是使用kinit来完成的，kinit将username传给AS，AS查找username的密码，将TGT和SK\_TGS使用用户密码加密后发送给kinit，kinit要求用户输入密码，解密后得到TGT和SK；其中，TGT使用TGS的密码加密，信息内容为{ user, address, tgs\_name, start\_time, lisftime, SK\_TGS} K\_TGS

2. User向KDC中的TGS请求访问某个Service的ST，发送[ TGT, Authenticator ]；其中，Authenticator用于验证发送该请求的user就是TGT中所声明的user，内容为：{ user, addresss, start\_time, lifetime}；Authenticator使用的TGS和user之间的session key加密的，防止TGT被盗。TGS先使用自己的密码解开TGT获得它与user之间的session key，然后使用session key解密Authenticator，验证用户和有效期。

3. TGS判断无误后，为user和Service之间生成一个新的session key：SK\_Service；然后发送给user一个包：[ {SK\_Service} SK\_TGS, ST ]；其中，ST是使用Service的密码加密的，SK\_Service使用TGS和user之间的session key加密的；ST的内容为：{ user, address, start\_time, lifetime, SK\_Service } K\_Service

4. User使用与TGS之间的会话秘钥解开包得到与Service之间的会话秘钥SK\_Service，然后使用SK\_Service生成一个Authenticator，向Service发送[ ST, Authenticator ]；其中，此处的Authenticator是使用user和service之间的会话秘钥加密的，Service收到包后先使用自己的密码解密ST，或者会话秘钥SK\_Service，然后使用SK\_Service解密Authenticator来验证发送请求的用户就是票中所声明的用户。

5. Service向用户发送一个包以证明自己的身份，这个包使用SK\_Service加密。此后user与Service之间使用SK\_Service进行通信，且在TGT有效期内，user直接跳过第一步直接从第二步使用TGT向TGS证明自己的身份。注意：user client会等待service server发送确认信息，如果不是正确的service server，就无法解开ST，也就无法获得会话秘钥，从而避免用户使用错误的服务器。

## Kerberos的风险与缺陷

1. 单点失败
2. 时钟同步
3. 安全依赖
4. 集中式管理
5. krbtgt账户

# Kerberos使用操作说明

## 名词解释

1. **KDC：Key Distribution Center，密钥分配中心**，负责Kerberos密钥的分发和存储，KDC中自带一个数据库用于Realm和Principal等数据的存储
2. Principal：Principal是Kerberos中参与认证的基本实体，通常来说Principal的含义有两种，一种用于表示Kerberos数据库中的用户，一种用于表示一台特定的主机，可以认为**Principal是代表了用户或主机的身份信息**。
3. Principal的命名格式：通常Principal的命名格式为Name/Instance@Realm。其中Name代表了用户名或主机名，Instance是对Name指代用户或主机的进一步描述，可以是用户所在的主机或类型等信息。Instance可以省略，在未省略的情况下需要与Name使用/号隔开。
4. **Principal的认证：Kerberos中支持的用户认证方式有两种，一是由用户输入密码进行认证，二是将Principal导出至Kertab文件后使用Kertab文件免密码认证**。对于一个Principal来说只能选择其中一种方式进行认证，由于Hadoop及其各组件中使用的机制原因，对于服务端的Kerberos认证只能使用Keytab文件进行认证。
5. **Realm：作用域，指认证系统所管理的范围，用于指定认证服务所影响的用户、主机以及服务的边界**。用户与服务并不一定要属于同一个Realm，如果双方所属不同的Realm，并且这两个Realm直接存在信赖关系，就能够实现交叉认证。一般来说，当一个用户或服务属于一个Realm时，它与认证服务器直接存在一个共享的Realm密钥。Realm的名称也是KDC中存储Principal的域数据库名。Realm在使用时对字母的大小写敏感，因此一般使用大写字母来区分，同时建议使用网路环境的DNS作为Realm。在输入Principal时，Realm可以省略，省略后默认使用本地的Realm。在未省略时使用 @ 号与Name或Instance分隔
6. **Keytab：保存了一个或多个Principal及其信息的密钥文件**，由KDC管理员导出生成，用户可以直接使用该文件进行免密码认证，因此需要注意密钥文件的文件权限。

## KDC常用操作

## Client常用操作

# Kerberos集成环境配置

1. 集群环境所使用的组件版本



1. 系统用户名和用户组



1. 主机名

由于Hadoop会将所有的主机名转化为小写字母执行，而Kerberos对大小写子母敏感，因此集群中的所有主机名必须为小写字母才能正常运行。

1. 集群中各节点角色及服务说明



1. 路径说明

Hadoop组件目录均放置于/var/local/hadoop/路径下

1. Realm说明

所使用的Kerberos Realm均使用LINC.COM表示

## Kerberos服务配置

1. 安装KDC
2. 安装Kerberos客户端

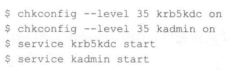
Kerberos的客户端需要安装在所有需要使用Kerberos协议进行加密传输的主机上，包括用户所使用的主机

1. 配置Kerberos服务端
2. 创建Kerberos数据库

为了存储之后需要用到的Principal，需要在linc-krb上以域名创建KDC中的数据库。



1. 启动Kerberos服务



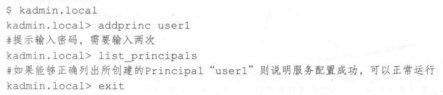
1. 创建Kerberos管理员

为了管理KDC中存储的Principal，需要在KDC上为Kerberos建立管理员账号，在linc-krb上使用root账号执行以下命令：



1. 测试Kerberos服务

为了确认Kerberos服务端的配置成功，可以尝试在KDC上以root用户执行以下命令进行测试：



1. 配置Kerberos客户端

Kerberos客户端通过修改配置文件进行配置，客户端的配置文件通常存放于/etc/krb5.conf，需要在所有安装了Kerberos客户端的主机上使用root用户对其进行修改配置。

1. 测试Kerberos认证

要测试Kerberos认证，需要在KDC中创建新的Principal，并将其导出为Kertab文件分发至客户端，并在客户端使用该Keytab进行Kertab认证。





## HDFS集成Kerberos的安装与调试

通过Hadoop HDFS服务与Kerberos的集成配置，实现以下功能：

1. 没有经过Kerberos认证的用户无法使用HDFS服务以及Web UI；
2. 经过Kerberos认证的用户可以正常使用HDFS服务以及Web UI；
3. 编译环境的安装配置
4. 安装maven
5. 安装编译所需的依赖包
6. 安装Protobuf

Protobuf，即Protocol Buffer，是Google开源的一种轻便高效的结构化数据存储格式，可以用于结构化数据序列化和反序列化，很适合用于数据存储或RPC数据交换格式。hadoop使用了Protocol Buffer作为RPC的默认实现，因此在编译Hadoop时也需要使用到Protocol Buffer。

1. 编译hadoop
2. 下载hadoop源码
3. 编译
4. 获取编译后的Hadoop
5. 配置Hadoop
6. 创建HDFS的Principal
7. 合并Keytab
8. 分发Keytab文件
9. 修改Hadoop配置文件
10. 配置JSVC

JSVC是Java应用的辅助程序，其中包含了一定数量的库和应用程序，能够使一些Java程序使用root权限进行一些需要权限的操作，并进行用户之间的切换。

由于HDFS中的DataNode在配置了Kerberos后需要root权限进行操作，因此需要安装JSVC来辅助运行。

1. 下载并解压JSVC
2. 编译生成JSVC
3. 分发JSVC
4. 分发JSVC依赖包
5. 修改Hadoop配置文件

修改hadoop-env.sh文件



1. 启动HDFS
2. 测试HDFS
3. 测试HDFS Web UI

Hadoop Web UI在开启了Kerberos验证后也同样需要Kerberos的票据才能访问，由于目前主流浏览器不会自动将Kerberos的票据提交至Web页面，因此需要针对不同的浏览器做出不同的调整。以Firefox为例：

1. 打开Firefox浏览器，在地址栏中输入about：config，进入设置界面；
2. 在顶部的search栏中输入network。negotiate-auth.trusted-uris搜索该配置项
3. 双击行，在对话框中输入需要浏览器提交票据的url，可以使用逗号分隔多个url

配置完成后，先在没有认证的情况下，访问NameNode的Web UI，默认端口为50070，会显示 Http Error 401 authentication required。

进入Terminal， 进行Kerberos认证：



再访问NameNode Web UI,即可正常访问。

## YARN集成Kerberos的安装与调试

1. 创建YARN服务使用的Principal
2. 合并Keytab
3. 分发Keytab
4. 修改YARN配置文件
5. 创建container-executor配置文件
6. 分发container-executor配置文件并设置权限
7. 创建hadoop-secret文件
8. 测试准备
9. 启动YARN服务
10. 启动JobHistory服务
11. 创建Principal相关用户
12. 测试提交MapReduce任务
13. Web UI测试

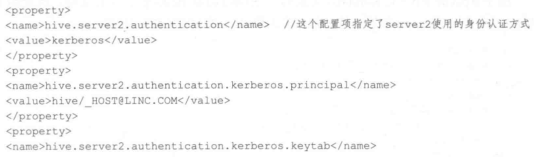
## Hive集成Kerberos的安装与调试

1. 修改Hive配置文件

Apache Hive开启server2服务，需要在配置文件中开启相应的配置项。修改hive-site.xml文件。



1. 创建Hive服务使用的Principal
2. 分发Keytab文件
3. 修改Hive配置文件





1. 修改相应Hadoop目录下core-site.xml配置文件
2. Hive Server2与metastore服务启动
3. 测试Hive metasotre服务
4. 测试Hive server2服务

## Zookeeper集成Kerberos的安装与调试

1. 创建Zookeeper服务使用的Principal
2. 分发keytab文件
3. 修改Zookeeper的zoo.cfg配置文件
4. 创建java.env文件
5. 创建jaas.conf文件
6. 启动Zookeeper服务

## Hbase集成Kerberos的安装与调试

1. 创建Hbase服务使用的Principal
2. 分发keytab文件
3. 修改Hbase配置文件
4. 创建zk-jaas.conf文件
5. 启动Hbase服务
6. 测试Hbase服务

## Sqoop集成Kerberos的安装与调试

1. 创建sqoop服务使用的Principal
2. 分发keytab文件
3. 创建sqoop.properties文件
4. 测试sqoop服务

## Hue集成Kerberos的安装与调试

1. 下载Hue源码
2. 安装依赖包
3. 编译Hue
4. 修改Hadoop配置文件
5. 修改Hue配置文件
6. 为Hue创建Principal
7. 分发Keytab
8. 修改profile文件
9. 开启Hue服务
10. 测试Hue连接HDFS和YARN
11. 在Mysql中创建hue用户
12. 开启HiveServer2
13. 添加Hive测试数据
14. 测试Hue连接Hive
15. 修改Hbase配置文件
16. 分发修改后的Hbase配置文件
17. 修改Hadoop配置文件
18. 分发修改后的Hadoop配置文件
19. 重启Hadoop服务
20. 为Hbase添加HTTP的Principal
21. 开启Hbase服务
22. 添加Hbase测试数据
23. 测试Hue连接Hbase

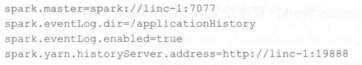
## Spark集成Kerberos的安装与调试

1. 创建Spark服务使用过的Principal
2. 分发Keytab文件
3. 配置Spark文件

spark-env.sh配置文件修改如下：



spark-default.conf配置文件指定了Spark运行时的日志保存位置。内容如下：



1. 启动spark
2. 测试spark服务

## Solr集成Kerberos的安装与调试

1. 下载并解压solr
2. 更改solr目录权限
3. 创建solr服务使用的Principal
4. 分发Keytab文件
5. 修改solr启动配置文件
6. 创建jaas-client.conf文件
7. 修改solr的配置文件
8. 在Zookeeper上创建solr的chroot目录
9. 在security.json中添加配置信息并上传到Zookeeper
10. 启动solr并创建collection
11. 测试使用solr

## Kafka集成Kerberos的安装与调试

1. 修改kafka server配置文件
2. 创建kafka server服务使用的Principal
3. 分发Keytab文件到各Kafka server端
4. 创建Kafka\_server-jaas.conf文件
5. 修改kafka Server端kafka\_run\_class.sh文件
6. 创建kafka Client端kafak\_client\_jaas.conf文件
7. 修改kafka Client消费者端console-consumer.properties文件
8. 修改kafka Client生产者端console-producer.properties文件
9. 修改kafak Client端kafka-run-class.sh文件
10. 启动kafka server
11. 在kafka server端创建topic
12. 测试kafka client（producer）
13. 测试kafka client（consumer）

## Storm集成Kerberos的安装与调试

1. 创建storm用户
2. 下载并解压缩storm文件
3. 配置storm文件
4. 配置Kerberos认证的storm
5. 配置Kerberos认证运行storm

## Impala集成Kerberos的安装与调试

1. 安装依赖
2. 创建Impala使用的Principal
3. 将生成的Impala.keytab分发至各个机器
4. 修改Impala.keytab文件权限
5. 修改Impala配置文件
6. 更新Impala配置文件下的文件
7. 启动Impala的state-store和catalog服务
8. 启动Impala-shell服务