# 根据Kerberos原理简单实现密钥分发管理系统

1.  **Kerberos原理简介**

1.1. **功能**

1. 一个安全认证协议
2. 用tickets验证
3. 避免本地保存密码和在互联网上传输密码
4. 包含一个可信任的第三方
5. 使用对称加密
6. 客户端与服务器（非KDC）之间能够相互验证
7. Kerberos只提供一种功能——在网络上安全的完成用户的身份验证。它并不提供授权功能或者审计功能。

1.2. **概念**

首次请求，三次通信方

* the Authentication Server
* the Ticket Granting Server
* the Service or host machine that you’re wanting access to.

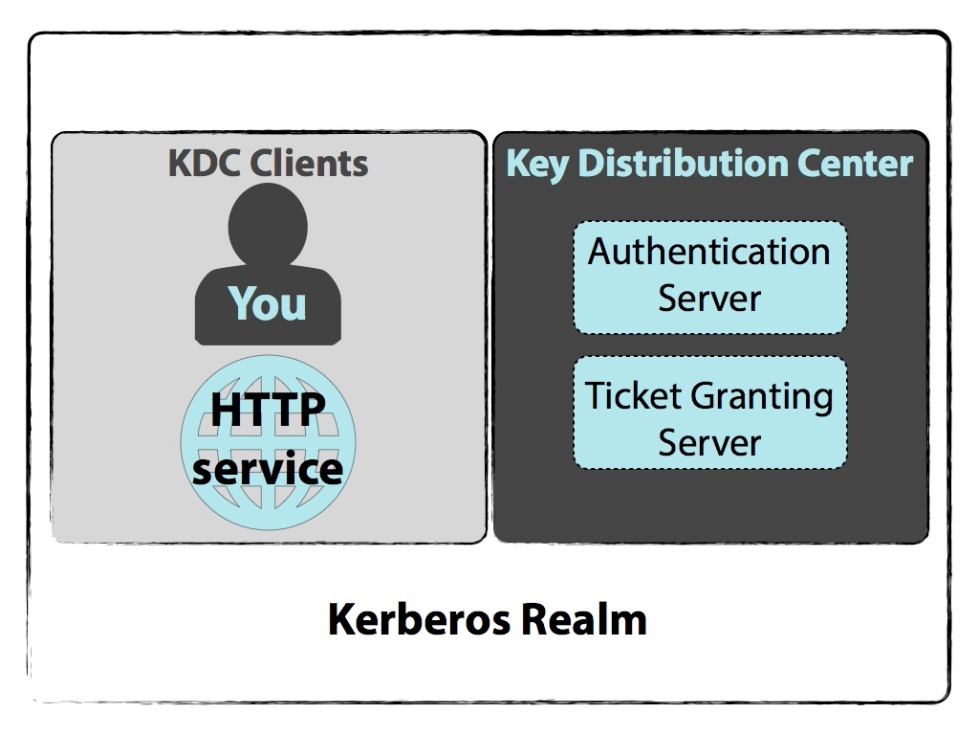


图 1‑1 角色

其他知识点

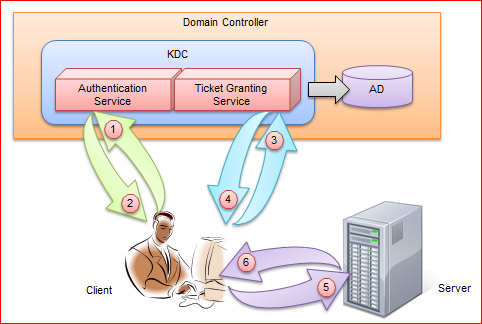
每次通信，消息包含两部分，一部分可解码，一部分不可解码

服务端不会直接有KDC通信

KDC保存所有机器的账户名和密码

KDC本身具有一个密码

2、  **3次通信**



　　我们这里已获取服务器中的一张表（数据）的服务以为，为一个http服务。

**2.1. 你和验证服务**

　　如果想要获取http服务，你首先要向KDC表名你自己的身份。这个过程可以在你的程序启动时进行。Kerberos可以通过kinit获取。介绍自己通过未加密的信息发送至KDC获取Ticket Granting Ticket (TGT)。

（1）**信息包含**

* 你的用户名/ID
* 你的IP地址
* TGT的有效时间

　　Authentication Server收到你的请求后，会去数据库中验证，你是否存在。注意，仅仅是验证是否存在，不会验证对错。

　　如果存在，Authentication Server会产生一个随机的Session key(可以是一个64位的字符串)。这个key用于你和Ticket Granting Server (TGS)之间通信。

（2）**回送信息**

　　Authentication Server同样会发送两部分信息给你，一部分信息为TGT，通过KDC自己的密码进行加密，包含：

* 你的name/ID
* TGS的name/ID
* 时间戳
* 你的IP地址
* TGT的生命周期
* TGS session key

另外一部分通过你的密码进行加密，包含的信息有

* TGS的name/ID
* 时间戳
* 生命周期
* TGS session key

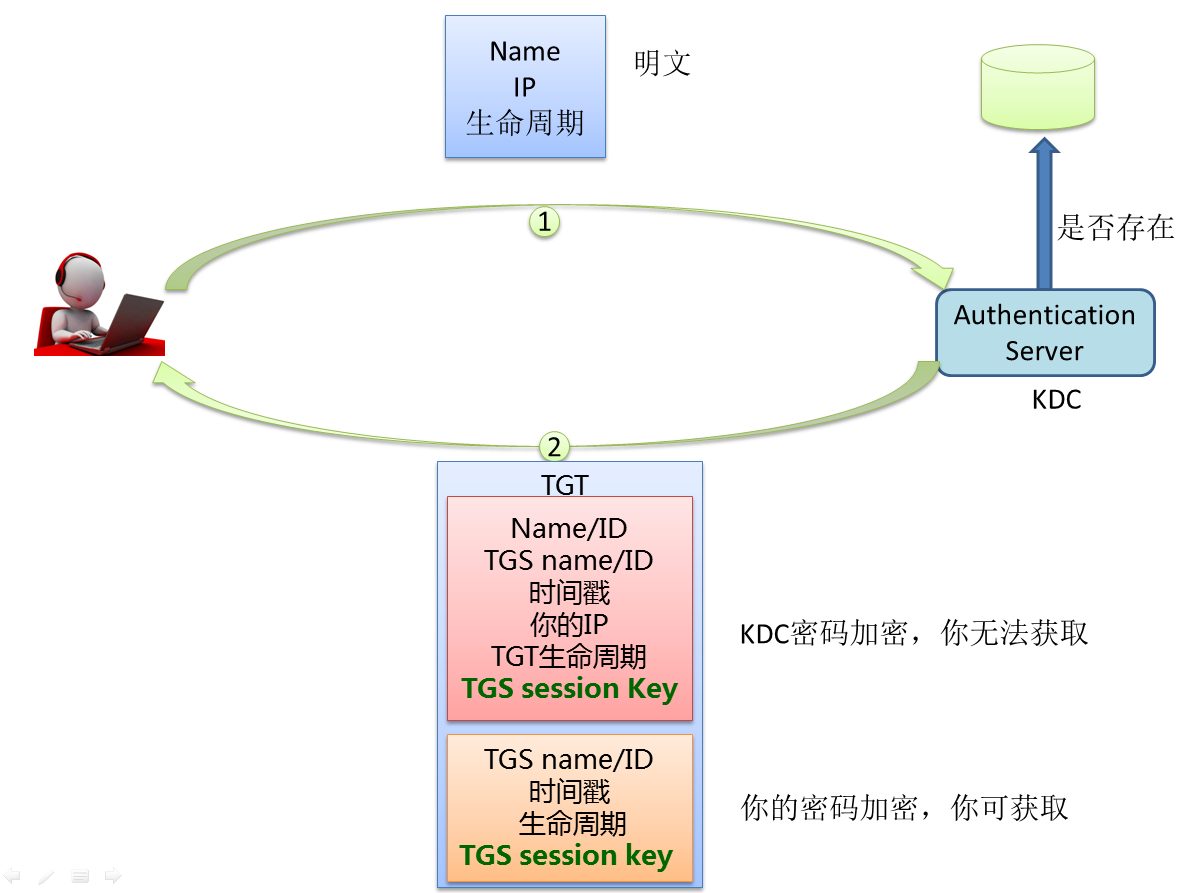


图 2‑1 第一次通信

　　如果你的密码是正确的，你就能解密第二部分信息，获取到TGS session key。如果，密码不正确，无法解密，则认证失败。第一部分信息TGT，你是无法解密的，但需要展示缓存起来。

**2.2. 你和TGS**

如果第一部分你已经成功，你已经拥有无法解密的TGT和一个TGS Session Key。

（1）    **请求信息**

1. 通过TGS Session Key加密的认证器部分：

你的name/ID

时间戳

1. 明文传输部分：

请求的Http服务名（就是请求信息）

HTTP Service的Ticket生命周期

1. TGT部分

　　 Ticket Granting Server收到信息后，首先检查数据库中是否包含有你请求的Http服务名。如果无，直接返回错误信息。

　　如果存在，则通过KDC的密码解密TGT，这个时候。我们就能获取到TGS Session key。然后，通过TGS Session key去解密你传输的第一部分认证器，获取到你的用户名和时间戳。

TGS再进行验证：

1. 对比TGT中的用户名与认证器中的用户名
2. 比较时间戳（网上有说认证器中的时间戳和TGT中的时间戳，个人觉得应该是认证器中的时间戳和系统的时间戳），不能超过一定范围
3. 检查是否过期
4. 检查IP地址是否一致
5. 检查认证器是否已在TGS缓存中（避免应答攻击）
6. 可以在这部分添加权限认证服务

　　TGS随机产生一个Http Service Session Key, 同时准备Http Service Ticket(ST)。

（2）    **回答信息**

　　a)        通过Http服务的密码进行加密的信息（ST）：

* 你的name/ID
* Http服务name/ID
* 你的IP地址
* 时间戳
* ST的生命周期
* Http Service Session Key

　　b)       通过TGS Session Key加密的信息

* + - Http服务name/ID
    - 时间戳
    - ST的生命周期
    - Http Service Session Key

　　你收到信息后，通过TGS Session Key解密，获取到了Http Service Session Key，但是你无法解密ST。

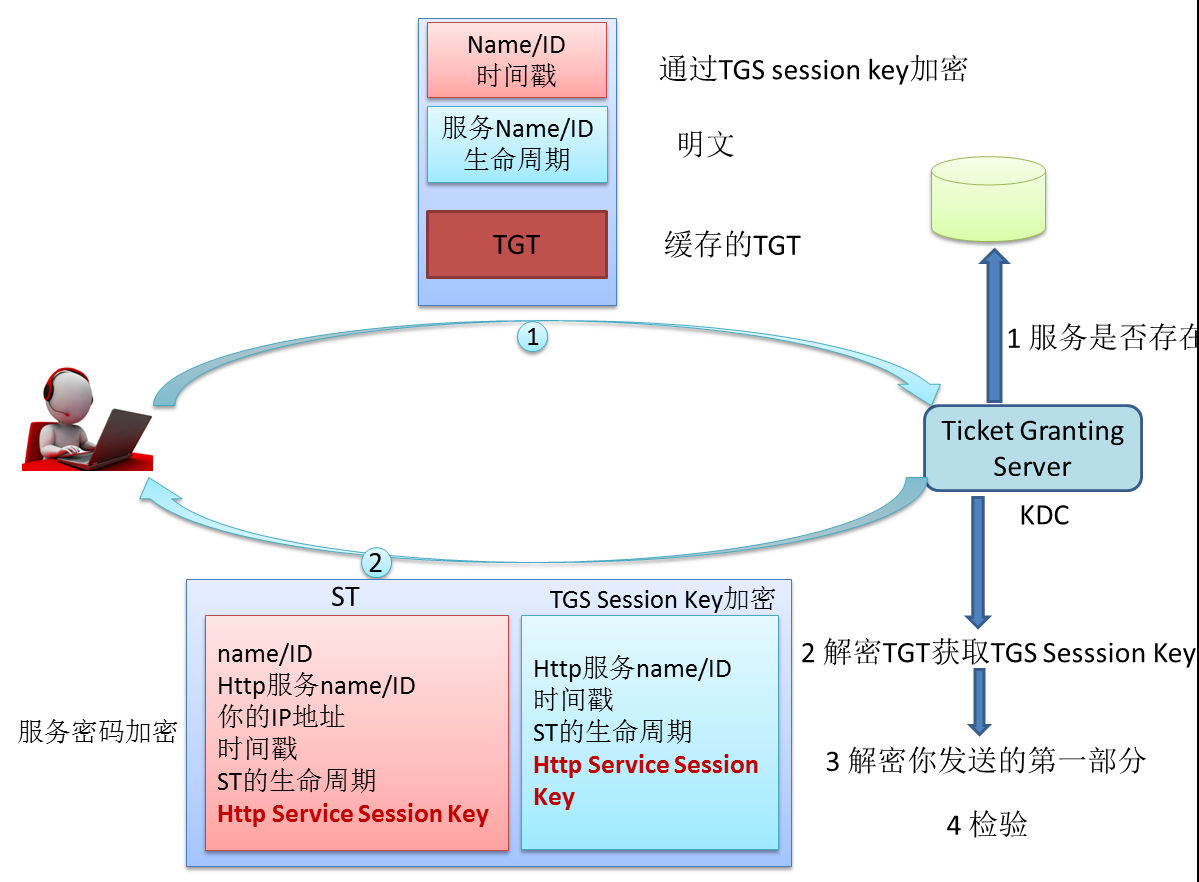


图 2‑2 第二次通信

2.3. **你和Http服务**

　　在前面两步成功后，以后每次获取Http服务，在Ticket没有过期，或者无更新的情况下，都可直接进行这一步。省略前面两个步骤。

（1）    请求信息

　　a)        通过Http Service Session Key，加密部分

* 你的name/ID
* 时间戳

　　b)       ST

　　　Http服务端通过自己的密码解压ST（KDC是用Http服务的密码加密的），这样就能够获取到Http Service Session Key，解密第一部分。

服务端解密好ST后，进行检查

1. 对比ST中的用户名（KDC给的）与认证器中的用户名
2. 比较时间戳（网上有说认证器中的时间错和TGT中的时间错，个人觉得应该是认证器中的时间戳和系统的时间戳），不能超过一定范围
3. 检查是否过期
4. 检查IP地址是否一致
5. 检查认证器是否已在HTTP服务端的缓存中（避免应答攻击）

（2）    应答信息

a)        通过Http Service Session Key加密的信息

* Http服务name/ID

时间戳

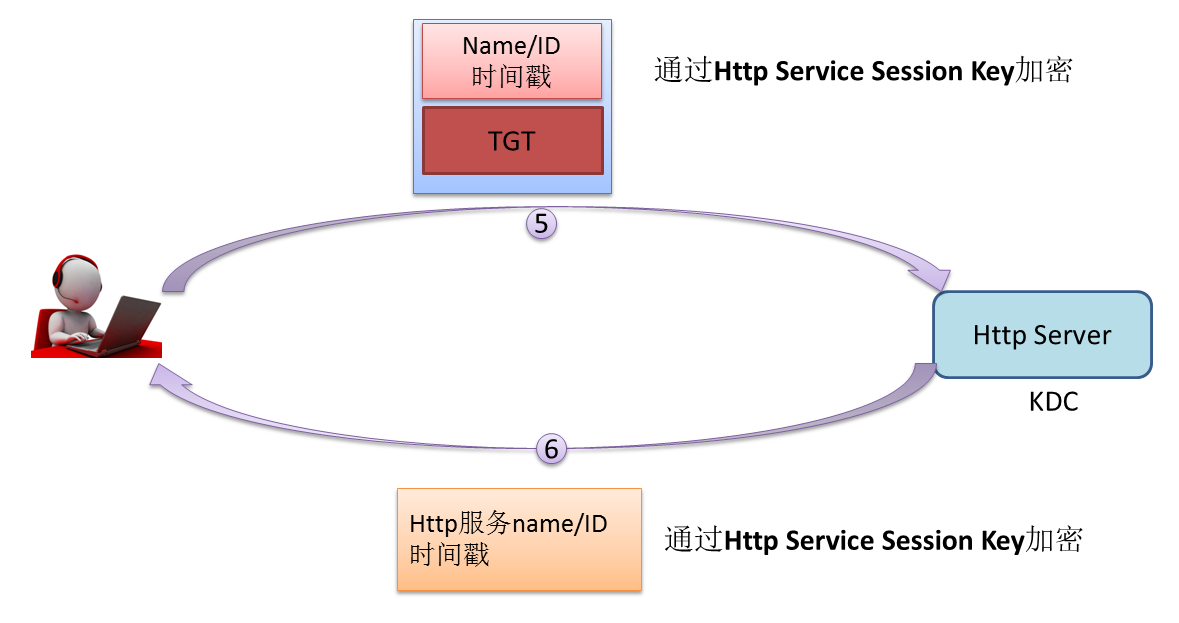


图 2‑3 第三次通信

　　你在通过缓存的Http Service Session Key解密这部分信息，然后验证是否是你想要的服务器发送给你的信息。完成你的服务器的验证。

至此，整个过程全部完成。

1. **实现**

现在，加密通过AES方式，密码保存用文件，序列化通过kryo.项目中使用后，准备再添加使用说明，和程序结构。

3.1.  **项目组成及功能**

|  |  |  |
| --- | --- | --- |
| ****子项目名**** | ****功能**** | ****依赖**** |
| ****ks-parent**** | Pom类型，定义依赖与版本 |  |
| ****ks-common**** | 传输的bean，异常体系，基本工具类 | ks-parent |
| ****ks-server**** | KDC服务端 | ks-parent、ks-common |
| ****ks-client**** | KDC客户端 | ks-parent、ks-common |
| ****ks-tool**** | KDC工具类，服务端database文件管理，客户端密码文件管理。可以扩展数据库 | ks-parent、ks-common |

3.2. **Ks-common**

　　基础的JavaBean，主要包含6次传输过程中的javabean，还有ServiceTicket与TicketGrantingTicket。

　　Kerberos部分错误通过异常进行传输，因此涉及到的所有异常较多。在该系统中所有的异常均为runtime异常，避免强制catch。

　　工具类，主要包含Kryo序列化工具（非线程安全）、安全工具类（AES、DES加密算法）、时间比较类。

3.2.1.    **基础类（com.xindun.kerberos.bean）**

这里的类均为2章中，3次通信过程中的信息封装。所有的类均是可序列化的，在该项目中通过Kryo进行序列化。

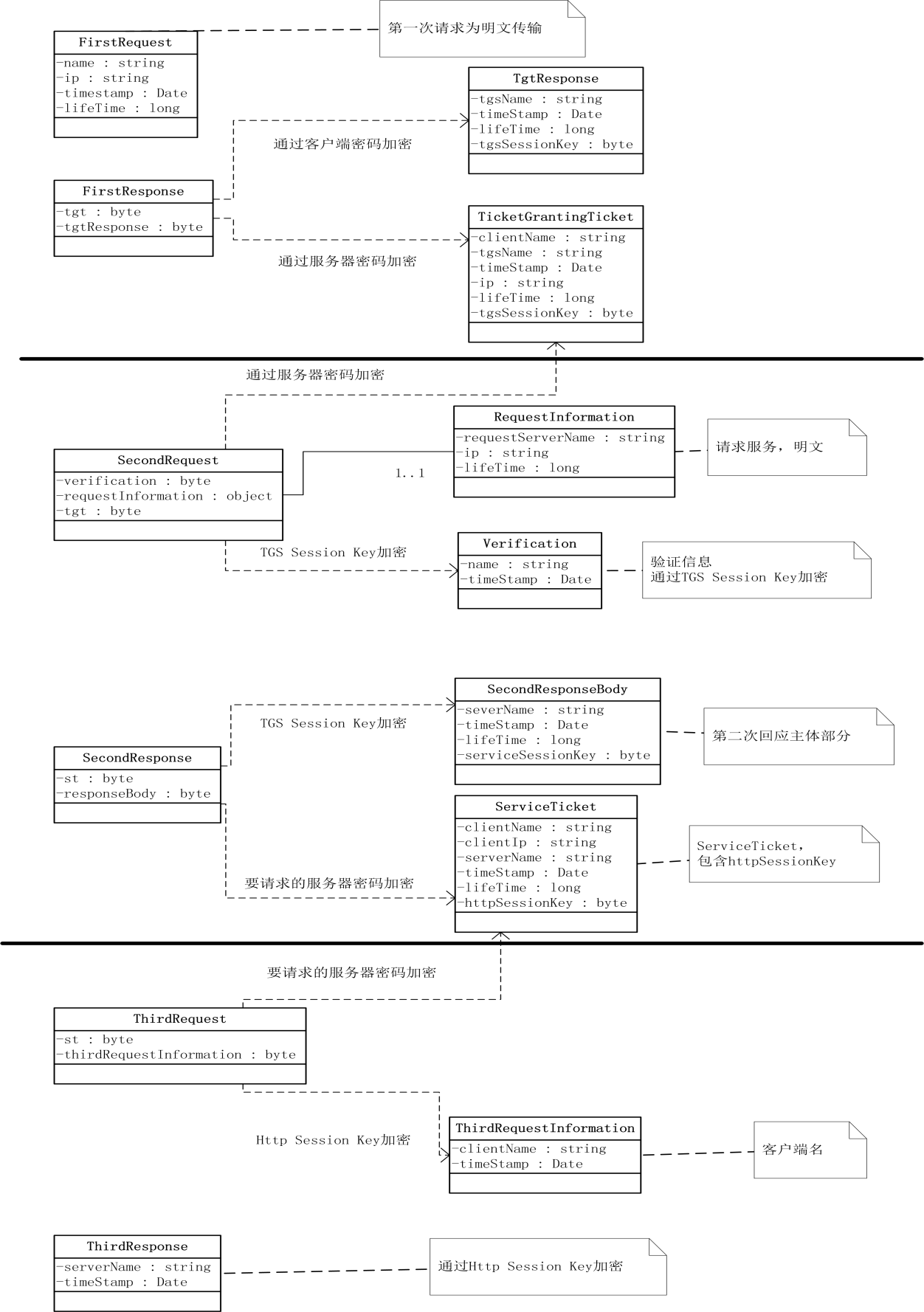


图 4‑1通信间的信息封装

除了第一次请求和最后一次应答，其他部分均包含能解析部分与不可解析部分。持有者不可能对不可解析部分修改，所以不可解析部分包含着进行验证的信息及用于解密的钥匙。

TGT包含的是客户端信息，及一把钥匙；主要用户KDC验证，请求者是否合法。ST包含客户端信息，及一把钥匙；主要用于请求服务器（非KDC），验证客户端。

1.2.2.    异常体系

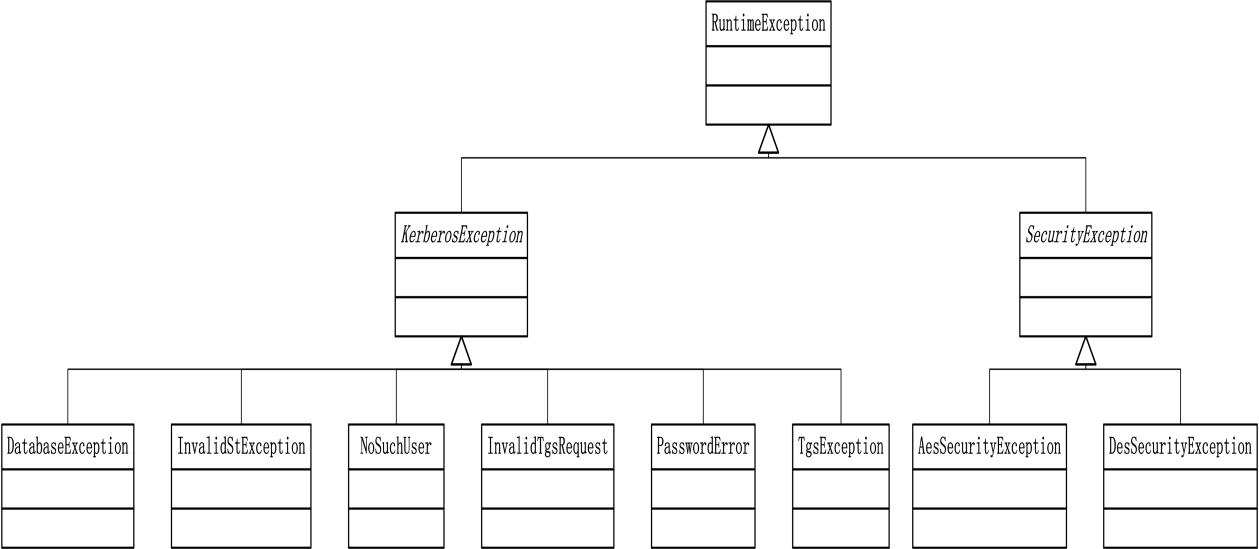


图 4‑2 异常体系

　　异常体系主要由Kerberos的异常系列和加密算法异常体系两部分组成。

3.2.3.    工具类

　　KryoSerializer：将对象转化为byte[]二进制，将二进制转化为对象。

　　Kryo在持久化速度和持久化后的空间上，都有无可比拟的速度。它的缺点：只能在java语言中使用、不能做兼容性。考虑到，调用该kerberos的系统使用的持久化方式为kryo，我们这里默认的也采用kryo持久化。Bug较多，不过在该系统使用到的功能中，均无bug。

　　在该框架中，不需要用到对象之间的引用。同时，因为是通过网络传输的，所以不能进行注册。（会把类的全限定么全部持久化进去）。

　　KryoSerializer据说是非线程安全的。

　　SecurityUtil：通过DES或者AES进行加密与解密。DES通过64位密码加密，所以如果字符串少于8个，则后面加0填充，如果多余则截取前面8位。AES通过128位加密，如果字符串少于16，则后面加0填充，多余则截取。

　　在该项目中，使用AES进行加解密。

3.3. Ks-server

　　KDC中心，主要包含Authentication Server、Ticket Granting Server。Authentication Server认证请求服务器是否合法（A请求B，KDC验证A的合法性），服务票据请求服务（Ticket Granting Server），授予服务请求票据（通过Ticket，B能确定A的合法性）。

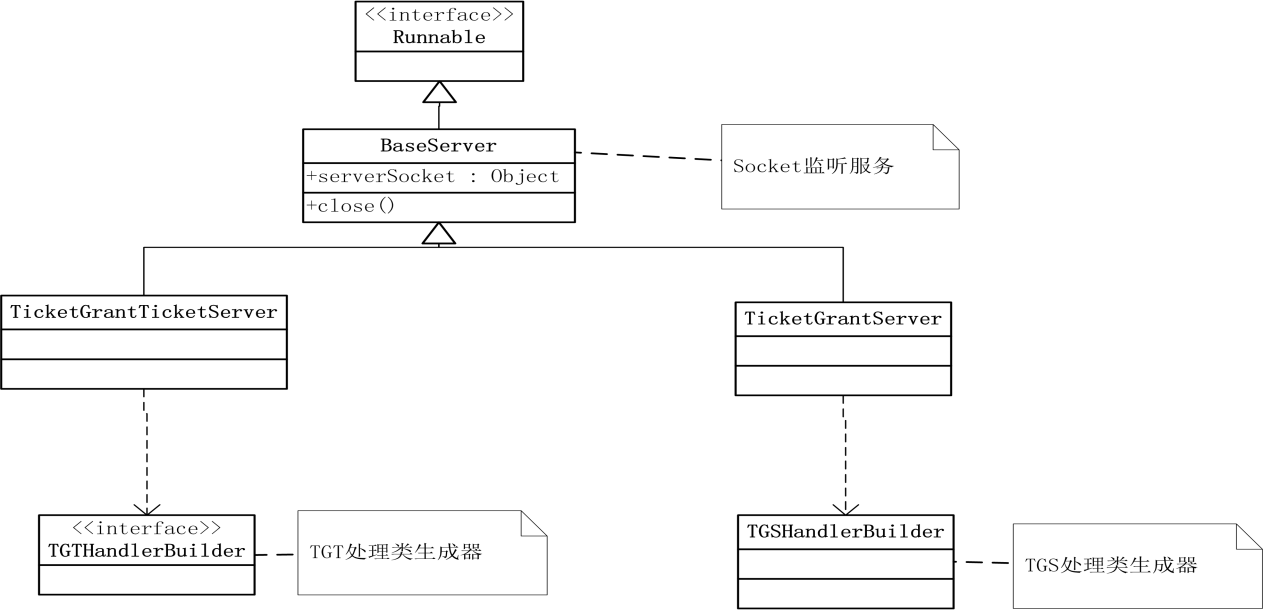


图 4‑3 KDC服务器监听服务

　　监听到Socket请求后，交由具体类进行处理。具体类由对应的对象生成器生成。通过对象生成器生成，可以方便的替换持久化方式、加密方式、数据保存方式（该系统不依赖于Spring）。

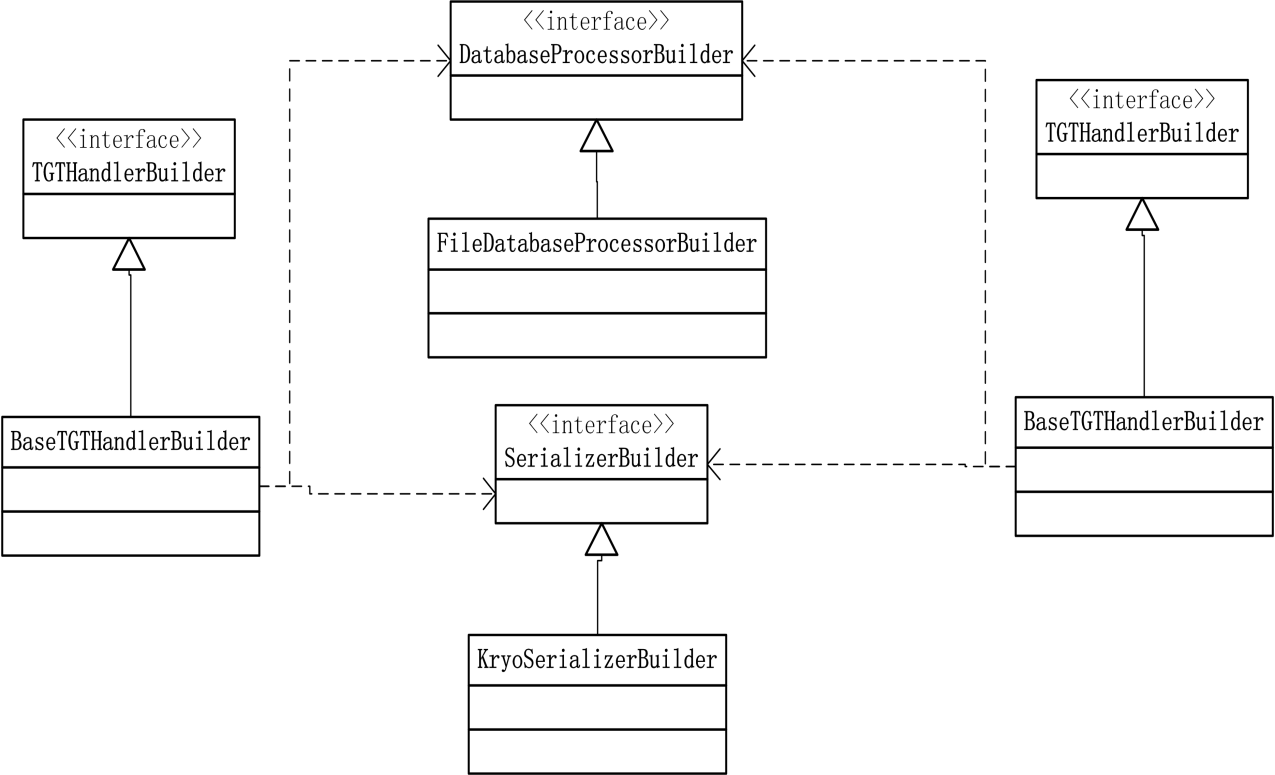


图 4‑4 生成器结构

　　通过构建者模式，生成复杂的类结构。TGT处理类生成器与TGS处理类生成器，均包含数据库处理器生成器与序列化生成器。数据库处理类生成器现在只实现了文件数据库生成器、以后一定会添加数据库数据库生成器（将用户名与密码保存至数据库）。序列化生成器，目前只包含Kryo序列化生成器。

3.3.1.    Authentication Server

　　BaseTgtProcessor类进行处理，服务端是无状态的，每次请求过来都不需要保存信息。FirstResponse check(FirstRequest firstRequest) throws NoSuchUser方法检查请求是否合法，并生成对应的应答信息。

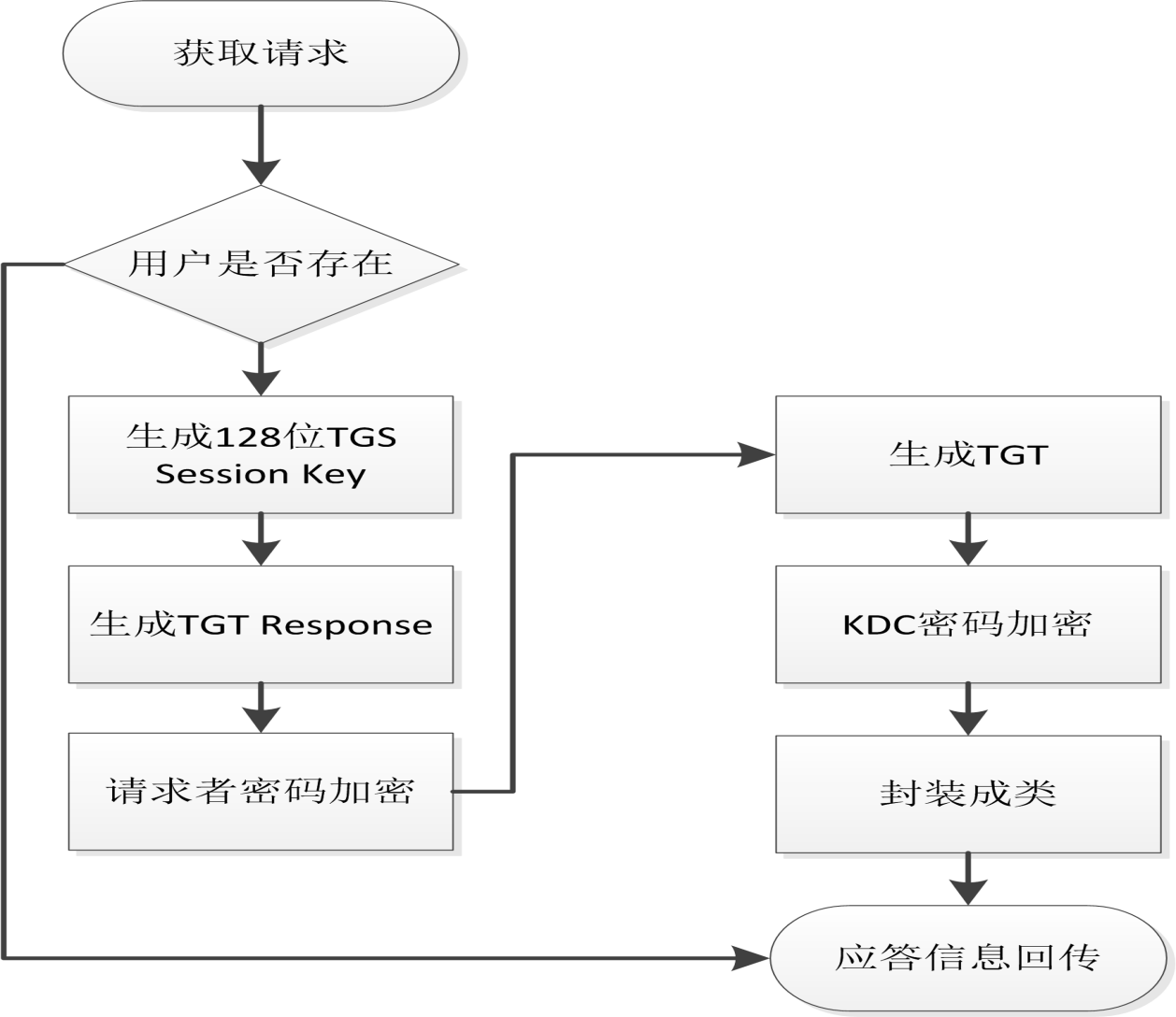


图 4‑5 检查合法性及应答信息生成

3.3.2.    Ticket Granting Server

　　服务票据授予服务，A请求B的服务。KDC给A一张票据，这样B能够确认A的合法性。

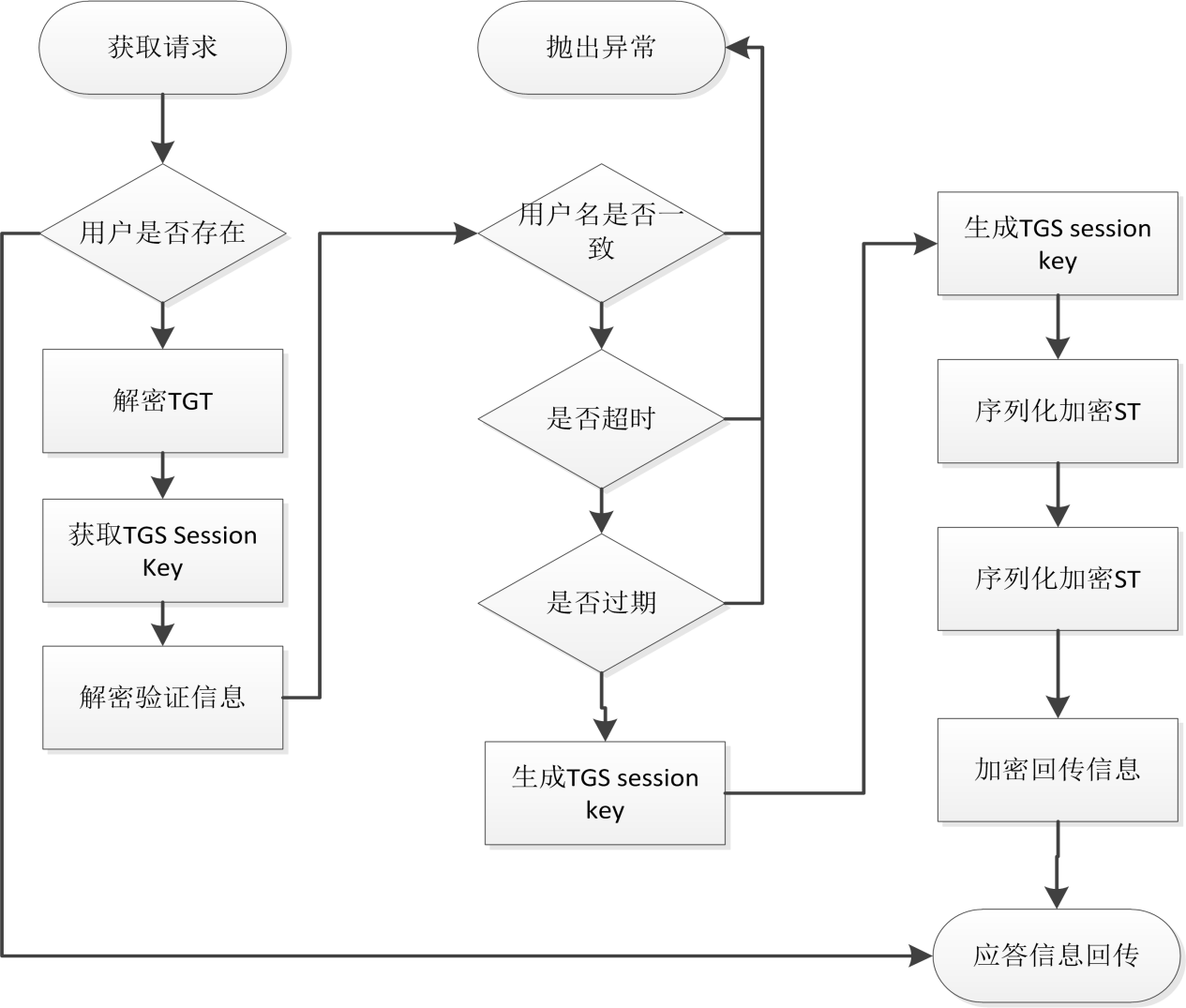


图 4‑6检查合法性及应答信息生成

3.3.3.    ServletContextListener

　　Servlet Context监听器，在tomcat启动的时候，也将KDC的服务启动起来。在关闭的时候，一起关闭，不需要单独成为一个应用程序。同时，能够在web.xml里面配置简单的参数。

3.4. Ks-client

　　KerberosClient接口中包含了，所有的基础方法。详见里面注释。客户端需要保存KDC返回的各类信息，JavaBean结构如下所示：

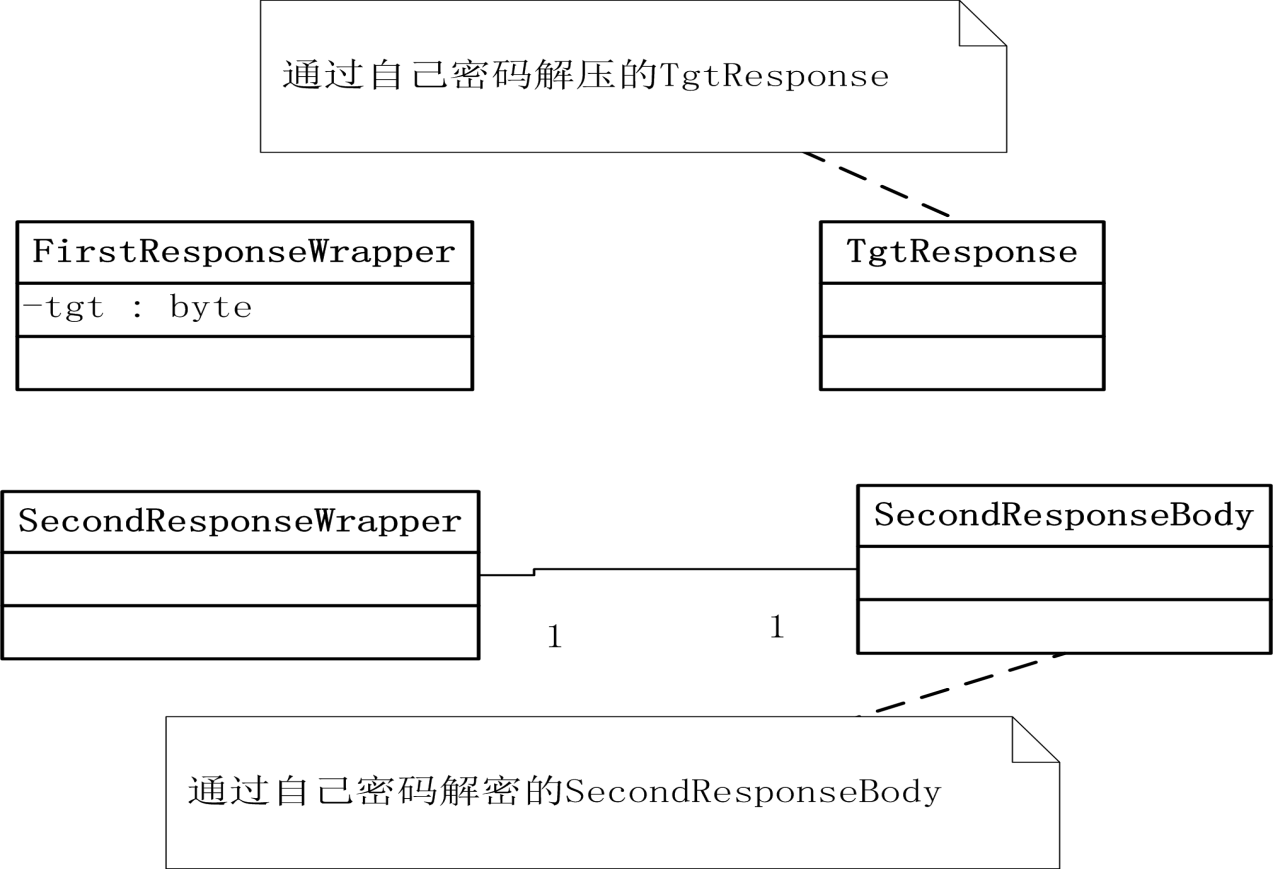


图 4‑7 客户端javabean

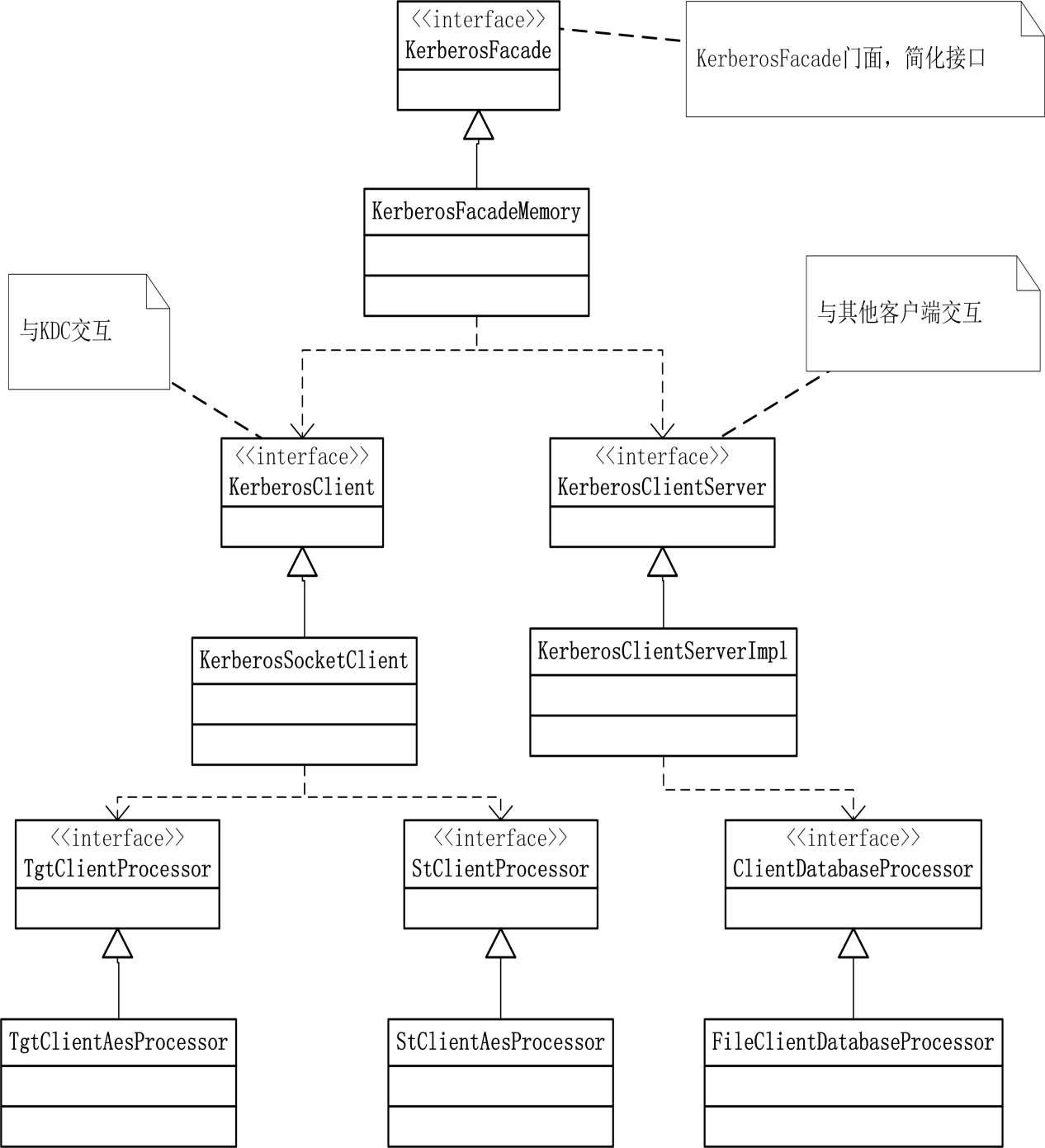


图 4‑8 客户端架构

　　KerberosFacade:门面模式，简化使用方式。1、获取请求服务时，如果ST或者TGT还不存在，会自动请求调用。2、其他客户端请求时，检查是否合法。3、请求服务返回时，检查是否合法。

　　KerberosFacadeMemory:将ST、TGT等信息保存在内存中。

　　KerberosClient:主要负责和KDC交互

　　KerberosClientServer:其他客户端交互，检查客户端是否合法，生成应答信息。

　　FileClientDatabaseProcessor:以文件形式，保存客户端的账户密码。

　　加密方式，KDC与客户端必须一致。我们这里默认使用AES加密，如果需要采用别的加密方式，需要重新实现。