

3AS概要设计文档

（仅供内部使用）

|  |  |  |
| --- | --- | --- |
| 文 件 编 号： | |  |
| 版 本 号： | |  |
| 实 施 日 期： | |  |
|  | | |
| 编 制： | 胡志云 | |
| 审 核： |  | |
| 会 签： |  | |
|  |  | |
|  |  | |
| 批 准： |  | |

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| 20110816 | 0.1 | 初稿 | 胡志云 |
| 20111117 | 1.0 | 评审修订 | 胡志云 |
| 20120823 | 1.1 | 维护文档，使文档与代码保持一致 | 胡志云 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 引言 5](#_Toc338854897)

[1.1 编写目的 5](#_Toc338854898)

[1.2 预期读者和阅读建议 5](#_Toc338854899)

[1.3 参考资料 5](#_Toc338854900)

[1.4 术语与缩略语 5](#_Toc338854901)

[2 设计概述 6](#_Toc338854902)

[2.1 限制和约束 6](#_Toc338854903)

[2.2 设计原则和设计要求 6](#_Toc338854904)

[3 软件架构 7](#_Toc338854905)

[3.1 系统边界 7](#_Toc338854906)

[3.2 总体架构 8](#_Toc338854907)

[3.3 线程设计 9](#_Toc338854908)

[4 模块功能 9](#_Toc338854909)

[4.1 CCtrlInst模块 9](#_Toc338854910)

[4.2 CNMSConnInst模块 9](#_Toc338854911)

[4.3 CLicenseInst模块 9](#_Toc338854912)

[4.4 CSyncAnalyzerInst模块 10](#_Toc338854913)

[4.5 DB作业池模块 10](#_Toc338854914)

[5 系统接口 10](#_Toc338854915)

[5.1 通信机制设计 10](#_Toc338854916)

[5.2 通信协议设计 10](#_Toc338854917)

[6 数据库设计 10](#_Toc338854918)

[7 业务设计 11](#_Toc338854919)

[7.1 启动与退出 11](#_Toc338854920)

[7.2 设备入网 12](#_Toc338854921)

[7.3 设备查询（分页查询） 13](#_Toc338854922)

[7.4 拓扑管理 13](#_Toc338854923)

[7.4.1 组网步骤 13](#_Toc338854924)

[7.4.2 退网步骤 14](#_Toc338854925)

[7.4.3 域名修改 15](#_Toc338854926)

[7.5 数据同步 15](#_Toc338854927)

[8 难点及处理 16](#_Toc338854928)

[8.1 消息头定义 16](#_Toc338854929)

[8.2 消息可扩展性 16](#_Toc338854930)

[8.3 消息防窜 16](#_Toc338854931)

[8.4 查询设计 16](#_Toc338854932)

[8.5 设备类型抽象 17](#_Toc338854933)

[8.6 会话管理 18](#_Toc338854934)

[8.7 数据多线程处理是否加锁 18](#_Toc338854935)

[8.8 数据操作是否支持事务性 19](#_Toc338854936)

[8.9 处理的有序性分析 19](#_Toc338854937)

[9 遗留问题 20](#_Toc338854938)

# 引言

## 编写目的

本文主要介绍了3AS模块的作用、定位和需要实现的功能，使初步介入的开发人员能够较快对3AS有整体认识与了解，更方便后续版本中3AS框架的优化与调整。

## 预期读者和阅读建议

项目经理和开发人员。

## 参考资料

公司10-Common\doc\design下面的产品设计文档

Andy Yue（美）,张玉祥,翟磊. 软件开发技能实训教程 技术文档篇. 科学技术出版社, 201012

## 术语与缩略语

1. AAA

Authentication、Authorization、Accounting ，认证、授权、计费。

“3AS”代表“authentication、authorization、accounting server”，“3AS”又表示为“TAS”（提出”TAS”的概念是因为C++代码名称不能以数字开头，’T’为”three”的第一个字母）。

1. DB

database的简称。

1. 3AC

3AS的客户端，全称为“authentication、authorization、accounting client”。目前的架构中pmcs会以3AC的身份登录3AS。

1. NMS

它是“network manage server”的简称，中文名称为“网管服务器”。目前的架构中pms充当NMS的角色。

1. 本级，上级，直属上级，下级，直属下级，顶级

3AS与3AS之间是存在层级关系的。

本级指的是当前平台自身；

上级指的是本级的父亲(一个节点只有一个父亲)以及父亲的父亲直至顶级；

直属上级指的是本级的父亲，也称为父亲平台；

下级指的是本级的儿子（一个节点可有多个儿子）以及儿子的儿子直至没有儿子的那个儿子；

直属下级指的是本级的儿子，也称为儿子平台；

顶级是没有父亲的平台。

1. 父亲，儿子

父亲与儿子就是数据结构中树的父亲节点与儿子节点。

1. 同步数据

这里的“同步”是个名词，“同步数据”指的本级平台需要向其它平台同步的数据，包括“平台关系”和“业务数据”。

“平台关系”是指平台的上下级关系信息，一个平台关系包括两个字段：平台3as ID，父亲平台3as ID。“平台关系”向上同步时保证“树”的完整性(也即同步上去的平台关系中的父亲平台3as ID在上级的平台关系列表中必须存在)，这样在上级可以看到所有级的层级关系，退网时可以做到直接删除相应的子树。

“平台关系”向上同步时如果上级发现已经存在相应的“平台关系”，但是“平台关系”中父亲平台不同，这时不允许下级进行同步（它需要等待里面的信息被删除）。

1. 同步路径

3AS之间数据同步存在两种路径，一种是“上下级之间的”，另一种是“点对点的”；前者要求自动向上级递归同步，后者不需要自动向上级递归同步。

1. 外设

指的是监控系统中的外设模块，比如TVS、UAS等。

# 设计概述

3AS的全称是authentication、authorization、accounting server。3AS维护监控系统中的设备数据和用户数据（包括用户信息以及用户的权限信息）以及实施license管理。

3AS在整个监控系统中起一个很重要的衔接作用。我们在每级平台上配置1个3AS，各级上3AS之间通过同步数据发生联系。3AS主要向其父亲同步数据，父亲再向它的父亲同步，依此类推，直至顶级。作为一种补充，3AS允许将一些设备同步给指定的3AS。

## 限制和约束

3AS系统对操作系统和数据库的支持要求分别如下：

操作系统：linux(比如CentOS、Redhat)；

数据库：mysql, sybase, oracle。

## 设计原则和设计要求

尽量复用已有组件进行设计。

遵守产品线命名规范。

# 软件架构

## 系统边界



图 3.1 单个3AS与其它模块的关系

图例说明：

1. 3AS与其它模块都是基于SIP进行通信。
2. 3AS与上级3AS是一定有通信的。
3. 3AS与其它3AS在配置了向它们同步设备的情况下才进行通信。

3AS业务上为3AC、CMU和CUI（CU通过它与3AS通信）提供服务；3AS与3AS之间主要向父亲平台同步数据，另外允许配置多个3AS进行点对点同步。

## 总体架构



图 3.2 3AS外部架构图

正如外部架构图中所示，3AS在域内为本域的3AC、CMU、CUI和外设提供服务，同时接受网管服务器的管理。域之间的3AS，本级与父亲是一定存在通信的，本级与其它3AS只有业务上需要同步数据才进行通信（比如临时授权行为就可能引发通信行为）。



图 3.3 3AS内部架构图

正如内部架构图中所示，3AS内部规划成5大模块，也即网管服务器接入模块(CNMSConnInst)、license管理模块（ClicenseInst）、同步分析模块（CSyncAnalyzerInst）、DB作业池（数据库操作真正的执行者）、中心处理模块（CCtrlInst）。

CCtrlInst利用SIP通道接收3AC、CMU、CUI、外设的请求，然后再利用SIP通道将应答返回给它们，另外3AS之间的通信也借助CCtrlInst的SIP通道。

## 线程设计

基础框架自身线程不描述。系统设计一个主线程，加上“图 3‑3 3AS内部的架构图”中每个模块对应一个线程（DB作业池除外，它对应一组线程）。

# 模块功能

下面简单介绍一下3AS内部各个模块的功能。

## CCtrlInst模块



图 4.1 CCtrlInst总体结构

作为3AS内部的中心App，负责会话的状态管理和消息的会话标识校验。它是外部SIP消息的统一入口，设计有统一的路由算法，也即请求消息是利用会话号找到相应的会话任务（登录请求除外，它用于建立会话任务），应答消息利用task号找到相应的任务（可能是会话任务也可能是处理具体业务的任务）；它的任务主动发起的内部请求是通过“传输类消息”来做路由的，传输类消息会额外携带些路由信息。

## CNMSConnInst模块

它以客户端身份登录网管服务器，负责与网管服务器通信，同时负责消息的编解码。

## CLicenseInst模块

该模块负责管理license信息，包括license信息查询和定时检测。 它同时公布一组license信息查询函数调用接口，方便DB作业池对license信息的获取需求。

## CSyncAnalyzerInst模块

同步分析模块，它管理着一系列同步分析器，同时对外提供同步分析拦截API。我们只需要在需要同步的地方加上同步分析拦截API的调用代码，后续是否确实需要同步以及需要向哪些TAS同步，就全部由同步分析模块负责了。对于需要同步的数据记录，同步分析器负责将相应的同步记录索引信息保存至同步表中，以备同步发送器取来发送。

## DB作业池模块

DB（全称database）作业池负责实际的数据库操作，它里面设置一组工作线程，这样可以防止数据库操作导致的阻塞。对外它不保证FIFO（也即先到的请求先处理），如果碰到必须要求先后顺序的两条信令，上层自己在收到第一条的应答后再发送第二条请求。

对外它提供函数调用的请求接口，应答通过OSP消息发送至CCtrlInst。如果是SIP请求则应答消息中负责带回请求中的SIP消息头（也即TospSipMsgHead）。

# 系统接口

## 通信机制设计

3AS模块之间总共采用了4种通信机制，下面以表格的形式描述。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模块/机制 | osp通信 | 网管通信 | SIP通信 | 函数调用 | 备注 |
| CCtrlInst | 对内使用 |  | 对外使用 |  |  |
| CNMSConnect | 对内 | 对外 |  |  |  |
| CLicenseInst | 使用 |  |  | license信息获取API |  |
| CSyncAnalyzerInst | 对内 |  |  | 同步分析API |  |
| DB作业池 | 返回应答 |  |  | 接收请求 |  |

## 通信协议设计

参见3AS外部消息定义文件, tas\_event.h。

参见3AS内部消息定义文件, inner\_msg.h。

# 数据库设计

参见数据库设计文档。

# 业务设计

下面记录一些有代表性的关键业务设计。

## 启动与退出



图 7.1启动与退出流程图

## 设备入网



图 7.2 设备入网时序图

## 设备查询（分页查询）



图 7.3 设备查询（分页查询）时序图

## 拓扑管理

平台组网总体控制通过PMS进行(它利用网管通道配置相应模块)，下面仅考虑3AS内部的业务逻辑。

### 组网步骤

**组网操作是站在本级（也即将来的上级平台）的角度来进行的，步骤如下：**

1. 在儿子平台设置父亲的3as信息；

2. 在本级平台添加儿子的3as信息。

说明，上面的3AS信息至少包括域名和ID等信息。

**行为约定(本级)：**

* 登录验证

对下级登录的3as进行验证，不是儿子不让登录。

* 递归同步平台关系

收到儿子发过来的“平台关系”同步时，需要校验是否允许接收新的平台关系，如果不允许接收则返回失败。

合法的平台关系要求满足：第一，平台关系中的上级信息在本级中存在，第二，平台关系在本级中不存在不一致的数据（比如存在一个平台关系，但是它的上级信息不同）。

* 递归同步业务数据

收到下级同步上来的同步数据时自动再向上级进行同步。

**行为约定(儿子平台)：**

3as设置新的父亲时，自动向上级同步数据。向上同步“业务数据”时，确保它所属平台的“平台关系” 先同步上去，否则相应“业务数据”不往上同步。

**异常场景：**

1. 离线组网（指单方面在上级添加儿子）

解决：这时需要手工设置下级的父亲的平台3as ID。

1. 新添加的下级平台之前已经在另一个分支处加入了系统（根据先到与后到顶级又可以分两种情况）。

解决：在同步至顶级的过程中一定会在某级发现“平台关系”校验冲突的，这时需要用户在另一个分支上先手工退网，然后这边最终才能够完成同步。我们要求检测到冲突的平台发出告警通知用户。

1. 新添加的下级平台，之前退网时在系统中仍然存有残留数据（有可能是中间网络不通或者中间平台繁忙，暂时无法同步上去）。

解决：在同步至顶级的过程中一定会在某级发现“平台关系”校验冲突的，这时待网络恢复或者平台稍微空闲时，问题将自动解决。我们要求检测到冲突的平台发出告警通知用户。

### 退网步骤

**退网操作是站在本级的角度来进行的，步骤如下：**

1. 在本级删除儿子平台的3as ID；

2. 在儿子平台设置父亲为空。

**行为约定(上级)：**

删除儿子子树上相应的资源，并且向上同步删除相应子树的资源。

**行为约定(下级)：**

设置父亲为空即可。

**异常场景：**

1. 离线退网(指单方面在上级删除儿子)

解决：这时需要手工设置下级的父亲为空。

### 域名修改

域名修改不影响拓扑结构。对于TAS涉及修改的地方有：所有的儿子平台中父域的配置，点对点同步源域中同步配置的目标域信息和被同步数据的平台中本域域信息。修改流程为先修改本域的所有注册上来的儿子平台的父平台配置信息，然后修改点对点同步源域中同步配置的目标域信息，然后再修改本域的域名信息，并且自动触发同步流程。

这里修改儿子平台中父域的配置和点对点同步源域中同步配置的目标域信息是本域自动进行的，如果自动修改过程中出现了失败的情况或者当时存在不在线，那我们需要手工修改它们的父域配置。

附：我们利用的同步流程异常告警可以发现未正确的域名修改。

## 数据同步

数据同步指的是3AS与3AS之间数据传递的一个过程，具体的场景包括上下级之间的数据传递（称之为“上下级同步”）和任意两个3AS之间的数据传递（称之为“点对点同步”）。在上下级同步过程中接收数据的直属上级在收到数据和保存之后需要继续向它的上级进行数据传递，它的上级在收到数据的行为仍然与它一样，直至顶级平台(顶级保存传递过来的数据即可)；而在点对点同步过程中接收数据的3AS只需要保存下来即可，不再向上级同步。

数据同步系统由拦截点布置、同步数据分析器、同步数据发送器和同步数据接收器组成。拦截点布置就是在可能引起数据同步需求的地方添加同步分析API的代码；同步数据分析器具体分析哪些数据需要同步和保存具体的同步数据索引记录，同步数据分析器分为三大类（也即CSyncAnalyzerParent、CSyncAnalyzerP2PNoLimit、CSyncAnalyzerP2PLimit），它们有着共同的基类（CSyncAnalyzerBase）；同步数据发送器负责同步数据的构造与发送，实现为一个任务类，依附在具体的会话任务下面；同步数据接收器负责同步数据的接收与保存，实现为一个任务类，依附在具体的会话任务下面。

实现时由3AS具体业务代码保证放到同步队列中的数据顺序是正确的，然后同步代码保证同步时FIFO（先进先出）。同步数据表针对每个目的域分别设置一个，因为他们之间的同步需要并发推进，互不影响。

数据同步时数据的依赖关系处理很关键，具体处理细节参见《同步系统业务分析.xlsx》。



补充：全联通方案

由顶级往下面各个平台进行数据同步，这样就可以做到各个平台都具有所有的公有设备数据了。为了解决下面平台数目过多时顶级平台同步数据时的性能问题，我们可以在顶级平台上面再部署些假的上级平台，用它们进行数据同步，它们可以分工合作，每个假平台同步一些下面的平台。

# 难点及处理

## 消息头定义

这里消息头定义主要侧重外部消息的消息头，内部消息的消息头无此限制。

3AS外部消息的消息头分为两个，一个为请求消息的消息头（CEventReq），另一个为应答消息的消息头（CEventRsp）。消息头字段设置的详细意义就不说了，下面仅就会话ID和事务ID（或者称为序列号）的使用约定简单介绍一下。

会话ID服务于服务端，由它生成，客户端在整个会话中负责带回；服务端校验会话ID不对时直接丢弃相应消息并且回复一个会话号非法的错误码。事务ID服务于客户端（严格地说应该是事务发起者），由它生成，服务端在整个会话中负责带回；客户端可以利用它将相关消息放置在同一个事务中。

## 消息可扩展性

消息结构体变化描述：

1. 增加字段（此时老版本对接时就是减少字段了）
2. 扩展现有字段（扩展现有字段老版本上可能产生溢出）

扩展后兼容性要求：向前兼容，向后兼容。

我们外部的协议采用XML与struct互转的技术，增减字段可以轻松应对，不存在的字段采用默认值。扩展现有字段需要考虑溢出问题（因此扩展现有字段的方式建议采用增加字段来做到）。

问题1：对于扩展的字段与老版本对接时采用默认值的做法业务上如果出现不正确的现象怎么办？

答：碰到这种情况我们只能新增消息。

注意，暂时不考虑消息本身设置版本号（不兼容时增加版本号）的概念。

## 消息防窜

如何防止后面登录的客户端收到前面客户端遗留在3AS中的应答或者通知消息？

解决：3AS内部设置会话的概念，不同次的登录属于不同的会话，并且会话号生成要求在足够大的空间内采用循环使用的方式（而不是每次从最小的一个号开始）。

## 查询设计

数据库查询统一采用分页查询的实现方式，所有查询采用通用的查询基类，具体参见TAS中设备查询信令的基类。

实现思路：一个REQ + RSP + 一批 NTF（NtfReq+NtfRsp）的方式。

查询首页时服务端同时返回符合条件的记录总数，客户端后续可以根据记录总数分页查询后面的数据记录。为了加快查询进度，用户在查询出总数之后可以连续发几个查询页内容的请求。

客户端如果需要保存所有查询出来的结果，那么它需要处理如下的异常场景。

* 异常场景：

1. 在总数查出来之后，前面也查询了一些页，随后中间一些页中的数据被删除了，这个时候可能返回前1页中的内容且可能无法查询到指定的页数。
2. 在总数查出来之后，随后一些数据被添加了，它们如果添加在最后最多就是看不到，如果添加在中间，那也可能导致中间一些数据重复出现。
3. 在总数查出来之后，数据记录中查询条件相关字段内容修改了，这样可能导致数据记录的顺序被打乱，进而可能导致有些已经查询出来的记录在后续页码中又出现而未查询出来的记录可能跑到已经查询过的页码处了。

上面这些异常场景下不同页码的查询结果组装起来可能会不正确，为此在查询结果中引入了“数据源版本”的概念，前后查询如果数据源版本变化了那说明数据源的数据内容被修改了，此时整体查询（需要获取所有页码的数据）结果可能不正确，需要重新发起查询。

## 设备类型抽象



图 8.1设备类型能力集抽象方案图

图例说明：图中右上角小方框内的为实现时需要考虑的问题。

抽象的动机：细化设备划归精度（抽象之后可以很方便地针对能力集进行划归）和简化设备入网（抽象之后设备型号具体数值直接进行选择即可，不用重复输入）。

概述：3AS系统中具体的设备至少具有设备类型属性和设备型号属性。设备类型具有硬件能力集属性和划归属性，它们是在产品设计的时候确定下来的，不允许客户修改；设备型号属性是设备类型中硬件能力集数值的具体化和license数值，它可以在客户使用过程中被不断地丰富。

整个抽象过程中涉及硬件能力、设备类型、设备型号三个核心概念。硬件能力是对设备关键属性的抽象，设备类型由特定的硬件能力集合和硬件能力划归属性集合构成，设备型号是设备类型中硬件能力集数值的具体化和license数值。

## 会话管理

3AS为成功登录的3AS、CUI、PMCS或者CMU建立会话，内部以它们登录时上报的模块URI作为唯一标识，不允许重复登录。PMCS登录3AS时除了提供自身的模块URI之外，还需要提供3AC用户名和密码。

模块注销时3AS内部释放会话资源（会话上面绑定的资源主要就是会话号）。

3AS通过心跳检测来发现会话的异常终止。心跳检测都是针对模块登录时提供的URI来进行的，检测到会话终止就进行会话资源释放等操作。

## 数据多线程处理是否加锁

多线程操作数据时是否加锁的思考。

我们已经采用了如下机制：

1. 通过数据库关键字段或唯一索引，保证了KEY数据不会混乱
2. 不同的数据库操作线程处理类似于不允许重名的业务时会先进行是否重名校验

目前的实现是未加锁版本，如果出现重名，到时只能用户再手工改回去，这将冲突的概率大大缩小了；如果出现数据主键冲突则提示数据库操作失败，需要用户手工重做。

如果需要加锁，大体有两个地方可以加：

1. 在C++代码中加锁，针对不同业务设计不同的锁，如此增加、删除和修改时都需要用对相应的锁进行加解锁。锁的精度还要是行级而不能是表级。
2. 在数据库的SQL语句中加锁，由于业务逻辑足够复杂的应用使用脚本语言编写时复杂度会比C++高级语言高，故不采用。

结论：类似于重名等业务对原子性要求不特别强烈的场景，我们暂时保留类似于1.0的实现（也即不加锁），出现脏数据时手工改回或者服务端提示错误然后采用重做补偿的方式；个别字段要求原子性比较强的地方则采用主建，或上面提到的两种显式加锁的手段。

补充：C++代码加锁算法

以业务为单位设计锁（一个业务一个锁），一个锁上面锁住绑定业务正在操作的数据记录标识。涉及到影响相关数据时（比如删除一个用户，那它同时会影响用户设备划归业务）如何进行加锁？在被影响数据记录标识确定的场景加“行锁”，否则加“表锁”。

定义一个锁类，实现类似于“表锁”和“行锁”的功能，提供数据记录标识的情况下加“行锁”，否则加“表锁”。

“实体类”数据的Add, Del和Mod可以做到“行锁”，Add时没有数据记录标识，可以使用伪标识（比如线程号）；

“关联类”数据的Add,Del和Mod的自身操作都可以做到“行锁”，被级联Add, Del或Mod操作时可能无法精确到数据记录标识，这时就只能做到“行锁”了。

## 数据操作是否支持事务性

我们已经采用了如下机制：

1．一个操作涉及多张表时，主要采用严格设计几个表的操作顺序来做到“安全”。比如删除用户时，则我们先删除用户关联的数据（比如已经划归的设备），最后才删除用户自身的数据，这样即使期间出错了，也能够再次发起删除操作进行重做。

## 处理的有序性分析

1）不同类型操作针对同一数据记录时的乱序（也即先操作的后处理）的影响分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 先操作╲后操作 | Add | Del | Mod | Qry | 备注 |
| Add | NA | NA | NA | NA | 乱序时不可见，当然也无法操作了 |
| Del | NA | √ | √  最终一致（依赖于变更通知有序） | √ | 通过Id删除 |
| Mod | NA | √  最终一致 | √  最终一致（依赖于变更通知有序） | √ | 通过Id修改 |
| Qry | √ | √ | √ | √ | 最终一定能够看到一致内容的，重查或者变更通知机制保证 |

2）不同类型操作针对不同数据记录时的乱序（也即先操作的后处理）的影响分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 先操作╲后操作 | Add | Del | Mod | Qry | 备注 |
| Add | √ | √ | √ | √ |  |
| Del | √ | √ | √ | √ |  |
| Mod | √ | √ | √ | √ |  |
| Qry | √ | √ | √ | √ |  |

备注：上面的操作通过变更通知的有序性实现，最终都可以做到数据一致。

结论：处理的有序性没有要求，但是需要保证变更通知的有序性，变更通知无序时可能导致赃数据。

# 遗留问题

1. DB作业池与数据库之间如果出现性能问题，后面可以考虑在它们之间加一个cache层。因为3as程序与3as数据库是1对1的关系，因此加cache时还是比较容易保证cache与数据库内容的一致性。