【**系统管理平台**】

**接口设计说明书**

【SMP-IDD-V0.1】

拟 制

审 核

标准化

批 准

深圳市中航比特通讯技术有限公司

**修订历史记录**

**变更类型 A** - 增加 **M** - 修订 **D** - 删除

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 变更  版本号 | 日期 | 变更  类型 | 修改人 | 摘要 | 备注 |
| V0.1 | 2013.07.02 | A | 曹文科 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1 范围 5

1.1 标识 5

1.2 系统概述 5

1.3 文档概述 5

2 引用文档 6

3 接口设计 6

3.1 接口标识和接口图 6

3.2 I-IF1： 6

3.2.1 应用管理 6

3.2.2 客户端XmlSql操作信令 7

3.2.3 应用数据上报 8

3.3 I-IF2： 8

3.3.1 数据动态同步信令XmlDbSync (S-E-PIP & S-I-PIP) 8

3.3.2 数据传输信令XmlDbTransfer(S-I-PIP | S-E-PIP) 8

3.3.3 数据前转信令ForwardXmlSql(S-E-PIP) 9

3.4 I-IF3： 10

3.5 I-IF4： 10

3.5.1 数据存储总体方案 10

3.5.2 <Table> … </Table> 11

3.5.3 <Record> … </Record > 11

3.5.4 <Field/> 12

3.5.5 内建数据字典（~O~鸡与蛋的故事~O~） 12

3.5.6 操作日志 13

3.6 E-IF1： 13

3.4.1 客户端注册(G-VIP & S-E-VIP) 13

3.4.2 权限访问信令(S-E-VIP) 14

3.4.3 XML-SQL操作信令(S-E-VIP) 15

3.6.4 自发现管理 15

3.7 E-IF2： 16

3.7.1 客户端XmlSql操作信令 16

3.7.2 应用数据上报 16

4 模型设计 17

4.1 数据模型 17

4.1.1 表格化数据 17

4.1.2 XML模型抉择 18

4.1.3 表格模型描述 18

4.1.4 系统目录的规划 22

4.1.5 统一编码 23

4.1.6 XML-SQL 24

4.2 通信模型 27

4.2.1 组网模型 28

4.2.2 可靠通信 30

4.3 架构建议 45

4.3.1 系统管理服务器架构设计建议 45

4.3.2 单板系统管理模块架构设计建议 46

4.3.3 业务系统数据管理部件架构设计建议 46

4.3.4 客户端数据管理部件架构设计建议 47

5 需求的可追踪性 47

5.1 部署需求 47

5.2 数据配置功能需求 48

5.3 故障管理功能需求 48

5.4 业务管理功能需求 49

5.5 系统监控功能需求 49

5.6 系统信息上报功能需求 49

5.7 软件管理功能需求 49

5.8 系统安全功能需求 50

6 注释 50

# 范围

## 标识

适用系统：SMP：System Manage Platfram 系统管理平台 版本：V0.1

接口实体：

SMS：System Manage Server 系统管理服务器 版本V0.1

SMU：uni-board System Manage Unit，单板系统管理模块 版本V0.1

SMC：System Manage Client 系统管理客户端 版本V0.1

DBS：Database System数据库系统

FS：File System文件系统

AS：Application Server应用服务器

接口标识：

I-IF1：SMU与SMS之间的接口

I-IF2：SMS与SMS之间的接口

E-IF1：SMC与SMS之间的接口

E-IF2：AS与SMU之间的接口

I-IF3：SMS与DBS之间的接口

I-IF4：SMS与FS之间的接口

适用时：软件设计、软件实现、单元测试、集成测试、系统测试、合格性测试。

## 系统概述

系统管理平台系公司独立项目，为模块化通信平台的组成部分，主要为模块化通讯平台相关产品提供管理与维护功能，以方便用户对产品进行相关的管理与维护操作。

系统管理平台的目标是为模块化通信平台产品提供通用化的系统管理功能支撑，对产品中系统管理的功能需求进行抽象化，使用通用的逻辑进行处理，保持系统管理业务逻辑上的一致，以屏蔽不同产品在数据上的差异性。要做到在产品的具体实施过程中，只需要对系统管理平台做一些必要的配置，即可实现并满足产品对系统管理方面的功能需求。

系统管理平台需要有支持灵活部署的能力，以适配不同产品的不同部署需求。要能够适应简化的部署方式，也要能够适应多层级的部署方式。

系统管理平台对外提供统一功能接口，包括系统管理对客户端接口以及对业务应用接口。

系统管理平台主要包括以下功能：数据配置功能、故障管理功能、业务管理功能、系统监控功能、系统信息上报功能、软件管理功能、系统安全功能等。

## 文档概述

本文档用于指导系统管理平台及其客户端的部分模块设计，也可供系统管理平台系统测试人员系统测试进行参考。

本文档集中描述系统管理平台与客户端和应用系统之间的通信接口。

# 引用文档

SMP\_SSS\_V1.0 《系统管理平台—系统规格说明书》

SMP\_SSDD\_V0.1《系统管理平台—系统设计说明书》

# 接口设计

## 接口标识和接口图



|  |  |
| --- | --- |
| 接口名 | 接口描述 |
| I-IF1 | 单板系统管理模块和系统管理服务器之间接口 |
| I-IF2 | 系统管理服务器之间的通信接口，用于数据同步和前转工作。 |
| E-IF1 | 系统管理客户端与系统管理平台之间的接口 |
| E-IF2 | 应用系统与系统管理平台之间的接口 |

本章所描述的各种接口都依赖于第4章的模型设计。

### 节点编码

4.2.2.5.2基本信令定义中需要填写‘发端节点编号’：

(1)客户端在发送时，用自身的IP地址的uint32形式表示。

(2)系统管理服务器、单板模块在发送消息时，用0x00XXYYZZ，其中XX表示MDN编号，YY表示MUN，ZZ表示SSN，先定义系统管理服务器的SSN为0，单板模块的SSN为1，其它由应用自身定义。

(3) 应用在发送消息时，用0x00FFFFZZ表示，其中ZZ表示应用自身的SSN。大于1。

当最高8位为非零时，表示接入为客户端。

### 扩展模块

|  |  |  |
| --- | --- | --- |
| 模块号 | 模块描述 | 接口 |
| 2 | 客户端注册模块 | E-IF1，请参考3.4.1 |

### 扩展插件

扩展定义以下几个序列化插件：

|  |  |  |
| --- | --- | --- |
| 插件号 | 插件描述 | 接口 |
| 3 | 带路由信息的文件传输插件 | 参考3.4.4和3.2.4 |
| 4 | 前转插件 | 参考3.3.3 |
| 2 | 普通序列化插件 | 其它章节的信令插件 |

### 路由信息

目前涉及到需要前转的消息有XML-SQL、FTP、数据监控需要前转。

XML-SQL、FTP的请求来自客户端，目的地是某个单板，响应是从单板回到客户端。请求的路由因子用MMS表示（即MDN-MUN-SSN，当然如何理解MMS，完全可以有系统管理服务器决定）。响应的路由因子用ClientNode表示即可（即客户端IP地址）。

数据监控，是服务器发给客户端的，直接用ClientNode表示即可（即客户端IP地址）。

## I-IF1（单播接口）：

### 应用管理

#### 下载文件列表

基于序列化消息设计下载文件列表信令。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| GetDownLoadList | 8 | 应用模块发起，操作码为50，为的是为Ftp部分预留一些扩展空间。 |
| RetDownLoadList | 9 | 返回下载列表。 |

GetDownLoadList操作数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| MUN | Uint32 | 单板系统编码 |

RetDownLoadList操作数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| FileName | string | 文件名。 |
| MD5 | uint8[16] | 该文件的MD5编码。 |

最后一个空RetDownLoadList包表示文件列表下载结束，下载前，系统管理服务器需要执行XmlDb操作日志的刷新流程【参考3.5.6】，下载完成前，都需要锁住客户端的配置修改等操作。

**为减少MD5的计算量，需要下载的文件变更时自动计算MD5编码，尤其是Xml的数据文件。**

#### 文件下载

直接使用4.2.2.5.5所定义的信令。

### XmlSql信令

客户端请求Insert/Update/Delete是必须要发给单本模块的，对于Select却不一定，一般说来，在系统管理服务器就有了，但部分字段可能仅在应用中，那么就需要对部分字段的发起查询操作。

针对客户端XmlSql操作，我们定义两条序列化信令：ServerXmlSqlRequest，ServerXmlSqlResponse。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| ServerXmlSqlRequest | 10 | 服务端发起的XmlSql请求 |
| ServerXmlSqlResponse | 11 | 单板回应的XmlSql响应 |

ServerXmlSqlRequest用于客户端发起XmlSql操作请求，其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SSN | uint32 | 指示单板发给某应用 |
| ServerSession | uint32 | 该请求服务端的会话标识。 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete/Select四种。 |

ServerXmlSqlResponse用于客户端发起XmlSql操作请求，其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ServerSession | uint32 | ServerXmlSqlRequest.ServerSession |
| Result | string | <Result>…</Result>语句  该消息会多次返回，通过扩展Result节点属性Count来控制总记录数。 |

这里当某些Update请求，如果仅仅修改‘控制’型字段，可以不需要回应，那么可以考虑不做ServerSession，那么应用端无需给ServerXmlSqlResponse回应消息。

### 应用数据上报

这类情况主要针对故障、事件、状态的上报，落实到数据模型上讲，就是表格的修改，不过这类操作无需回应，因为应用无需等在这个响应上。

针对该类数据上报，扩展一个序列化操作：BoardXmlSqlNotice。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| BoardXmlSqlNotice | 12 | 单板发起的数据变更通知 |

BoardXmlSqlNotice的操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete三种。 |

### 文件传输信令

主要用于客户端操作单板文件，在标准的文件传输信令上**前插**MDN和ClientNode路由信息字段（功能上可直接利用基础文件传输信令），如：

FtpOpenReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientNode | uint32 | 指示那个客户端发出的。 |
| MDN | uint32 | 接收原始请求的MDN |
| RemoteFile | string | 远端文件名。 |
| Mode | uint32 | （1）0x01可写，0x02尾部读写，0x04清除[只有写的情况下清除选项才有效]。以上选项可以位或运算。  （2）如果是写文件，需要自动创建文件所在目录。  （3）能写就能读。 |
| ClientSession | uint32 | 本地会话标识，支持并发操作多个文件。 |
| Transfer | uint32 | （1）0=非传输；1=文件传输；为1时，Mode仅仅为0x01或0x02。  （2）使用无需响应的FtpWrite进行传输数据，直到之后一个空FtpWrite消息，并自动进行MD5计算，在FtpClose中进行校验。 |

1. 其它消息请参考4.2.2.5.5，并依次类推的加MDN和ClientNode路由字段。
2. 系统管理服务器不做会话管理。
3. 单板模块做会话管理。

#### 动态配置下载流程

对于配置数据，如果第一阶段，想做的很简单，由客户端生成配置文件，传递到系统管理服务器，系统管理服务器下载到单板，单板通知应用，告知某文件发生变更，应用自己读取即可。

对于上述情况，系统管理服务器可以判断出该文件是程序文件还是配置文件，如果是配置文件，在需要本地存储一份。

## I-IF2：

### 数据动态同步信令 (S-E-PIP & S-I-PIP，组播接口)

基于序列化消息，扩展数据同步信令XmlDbSync：

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| XmlDbSync | 6 | 序列化数据同步操作 |

XmlDbSync操作参数定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 | |
| XmlSql | string | 同步数据，这里全部为Xml文本数据。  仅支持Insert、Update、Delete语句。 |

注意，当Insert语句执行返回1时，需要执行修改操作。

内子网同步域SDN为0，外子网同步域SDN为1；

不同子网上同步的表格不同。

#### 文件同步流程

对于配置数据，如果第一阶段，想做的很简单，由客户端生成配置文件，传递到系统管理服务器，系统管理服务器下载到单板，单板通知应用，告知某文件发生变更，应用自己读取即可。

对于上述情况，系统管理服务器可以判断出该文件是程序文件还是配置文件，如果是配置文件，需要发起文件同步流程。

文件同步流程，基于基础的文件传输信令，仅取用FtpOpenReq、FtpWriteReq、FtpCloseReq三条消息即可（无需响应），并放置在组播通道上(不用建立服务端会话，需要建立客户端会话)。

### 数据传输信令XmlDbTransfer(S-I-PIP | S-E-PIP，单播接口)

系统管理服务器的数据管理部件，从本地加载完数据后，应该先从管理范围内的其它服务节点加载数据，如果本管理范围内没有其它服务节点，需要从外子网选择一个服务节点，加载全局数据。这才能保证数据的完整性。

在加载过程中，也会有动态同步XmlDbSync消息，此时需要在收端对XmlDbSync压队列【保持队列】。

数据传输采用独立的序列化单播信令进行，和数据同步的序列化信令独立开来。因为：

1. 数据传输只是一端发送，一端接收；数据同步可以是双向的。
2. 数据传输在短时间内会有大量的消息，如果和数据同步使用同一序列化队列，那么会致使数据同步发送窗口快速覆盖，而动态数据同步永久性丢包。
3. 数据传输是一对一的，为控制发端流量及可靠性，需要支持发送窗口机制；而数据同步是不需要的。

基于序列化消息，扩展数据传输信令：XmlDbTransfer

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| XmlDbTransfer | 7 | 数据传输信令 |

XmlDbTransfer操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| XmlSql | string | 仅考虑Insert语句，最后一个空语句表示数据传输结束。  当Insert语句执行返回1时，需要执行修改操作。 |

### 数据前转信令(S-E-PIP，单播接口)

目前仅针对XmlSql、Ftp、XmlSqlNotice有前转动作，采用前转插件。所有的前转工作只有解析和重新封装(并添加前转路由信息)，无需建立会话。

#### FTP前转

当客户端需要操作单板上的文件时，需要前转。前转的封装方式与上雷同，如ForwardFtpOpenReq

ForwardFtpOpenReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientNode | uint32 | 客户端节点号 |
| MDN | Uint8 | 接收原始请求的MDN |
| MUN | Uint8 | 目标单板的MUN |
| RemoteFile | string | 远端文件名。 |
| Mode | uint32 | （1）0x01可写，0x02尾部读写，0x04清除[只有写的情况下清除选项才有效]。以上选项可以位或运算。  （2）如果是写文件，需要自动创建文件所在目录。  （3）能写就能读。 |
| ClientSession | uint32 | 客户端会话标识，支持并发操作多个文件。 |
| Transfer | uint32 | （1）0=非传输；1=文件传输；为1时，Mode仅仅为0x01或0x02。  （2）使用无需响应的FtpWrite进行传输数据，直到之后一个空FtpWrite消息，并自动进行MD5计算，在FtpClose中进行校验。 |

1. 其它消息请参考4.2.2.5.5，并依次类推的加MDN和MUN路由字段。
2. 系统管理服务器不做会话管理。

#### XmlSql前转

针对ClientXmlSqlRequest，ClientXmlSqlRequest定义相应序列化前转信令：ForwardXmlSqlRequest，ForwardXmlSqlResponse

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| ForwardXmlSqlRequest | 50 | 前转XmlSql请求 |
| ForwardXmlSqlResponse | 51 | 前转XmlSql响应 |

ForwardXmlSqlRequest其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientNode | uint32 | 客户端节点号。 |
| MMS |  | 用作消息路由MDN-MUN-SSN |
| License | uint32 | 授权码 |
| ClientSession | uint32 | 客户端的请求会话编号。ClientXmlSqlRequest. ClientSession |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete/Select四种。 |

ForwardXmlSqlResponse其操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientNode | uint32 | 客户端节点号。 |
| MMS |  | 用作消息路由MDN-MUN-SSN |
| ClientSession | uint32 | ClientXmlSqlRequest. SessionId |
| Result | string | <Result>…</Result>语句  该消息会多次返回，通过扩展Result节点属性Count来控制总记录数。 |

中转的系统管理服务器节点，不建立会话，采用ClientNode来标识。

做会话管理的系统管理服务器，需要为会话标记是否为前转会话，前转会话中需要记录客户端节点号和连接的系统管理服务器的MDN，以便响应能正确返回。

#### XmlSqlNotice前转

针对XmlSqlNotice的前传定义定义相应序列化前转信令：ForwardXmlSqlNotice

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| ForwardXmlSqlNotice | 52 | 前转XmlSqlNotice请求 |

ForwardXmlSqlNotice用于系统管理服务器给客户端的XmlSql通知(没有响应)，其操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientNode | uint32 | 客户端节点号。 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete |

同一个变化，可能会有多个客户端在监控，那么需要发出多条XmlSqlNotice或ForwardXmlSqlNotice。通知或前转通知都不做会话管理。

### 令牌竞选与VIP

内子网竞选S-I-VIP和S-E-VIP；外子网竞选G-VIP。

S-I-VIP供单板连接使用，S-E-VIP供客户端连接使用，G-VIP供客户端首次连接使用。

## E-IF1（单播接口）：

### 客户端注册(G-VIP & S-E-VIP)

客户端通过G-VIP发送GetServersRequest给全局系统管理服务器，获取所有系统管理平台的E-VIP地址列表GetServersResponse。

客户端基于时间随机选择一个S-E-VIP，发送ClientLoginRequest给相应的活动系统管理服务器，登陆该系统管理平台。

系统管理平台将维护客户端的连接情况，并通过ClientLoginResponse。系统管理平台收到ClientLoginRequest时发现当前Client登录在其它系统管理平台上，那么需要更新登录关系。

为确保数据路由以及监控的实时通知，该连接关系通过一张内存表存储，并进行同步。

客户端无需保存E-VIP地址列表，当连接断开时，需重新发起GetServersRequest。

GetServersRequest、GetServersResponse、ClientLoginRequest、ClientLoginResponse都不是序列化消息，消息类型定义如下：（扩展Module=2，请参考3.1.2）

|  |  |  |
| --- | --- | --- |
| 消息 | 命令字(Cmd) | |
| GetServersRequest | 0 | 可选命令，如果客户端是直接配置访问的系统管理服务器的话，该信令可不考虑。 |
| GetServersResponse | 1 |
| ClientLoginRequest | 2 | |
| ClientLoginResponse | 3 | |

（1）GetServersRequest消息体(G-VIP)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| 空 | | |

（2）GetServersResponse消息体(G-VIP)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Count | Uint32 | 活动服务器数量。 |
| Servers | Uint32[] | 活动服务器的地址列表。 |

客户端发出GetServersRequest后，即进行串行等待。

（3）ClientLoginRequest消息体(E-VIP)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| UserName | string | 最大16字节，不包括结尾0。 |
| AuthCode | uint8[16] | 由UserName[16]+ UserPsw[10]+Time(yyyymmddhhmmss)生成的MD5编码。UserName和UserPsw不足位填0 |
| Time | uint | 登陆时间。 |

（4）ClientLoginResponse消息体(E-VIP)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Status | uint | 0=成功；1=用户不存在；2=鉴权失败。 |
| License | uint | 授权码；登陆成功后，会返回一个授权码。  该授权码在断链之前一直有效，**是否需要考虑动态授权码？如果需要，就要增加一条消息用于动态变更授权码。** |

客户端发出ClientLoginRequest后，即进行串行等待。登陆成功后，启动可靠的序列化服务。

### 权限访问信令(S-E-VIP)

基于序列化信令，设计权限访问消息：ClientGetRoleRequest和ClientGetRoleResponse

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| ClientGetRoleRequest | 1 | 角色查询请求 |
| ClientGetRoleResponse | 2 | 角色查询响应 |

ClientGetRoleRequest操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| License | uint32 | 授权码 |

ClientGetAbilityResponse操作参数定义

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | | 数据类型 | 参数说明 |
| Mode | | uint32 | 0=该结果为授权定义，1为禁权定义。  如果用户需要管理的对象较多，那就采取禁权方式定义，如果需要管理的对象较少，可以采取授权方式定义，为的是减少数据流量。 |
| TableCount | | uint32 | 授权(或禁权)表的数量数。 |
|  | TableName | string | 用句点分隔表示。 |
|  | Role | uint32 | 1=查询；2=插入、4=删除；8=修改；16=监控；可按位或。  根据Mode表示授权或禁权。  拥有监控权限，那么当该表格变化时，系统管理服务器会自动采用XmlSqlNotice通知到客户端。 |

### XmlSql & XmlSqlNotice信令(S-E-VIP)

基于序列化信令，XML数据操作信令ClientXmlSqlRequest、ClientXmlSqlResponse、XmlSqlNotice

系统管理服务器，对具体某客户端不支持并发的ClientXmlSqlRequest，对不同客户端需要支持并发ClientXmlSqlRequest。这个约束由客户端控制。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| ClientXmlSqlRequest | 3 | 客户端发起的XmlSql请求 |
| ClientXmlSqlResponse | 4 | 服务器发起的XmlSql响应 |
| XmlSqlNotice | 5 | 服务器发起的XmlSql通知 |

ClientXmlSqlRequest用于客户端发起XmlSql操作请求，其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| MMS | uint32 | 具体如何解析该参数，有系统管理服务器定义【可参考SSN -MDN-MUN这种方式】 |
| License | uint32 | 授权码 |
| ClientSession | uint32 | 客户端的请求会话编号。ClientXmlSqlRequest. ClientSession |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete/Select四种。 |

ClientXmlSqlResponse用于系统管理服务器给客户端XmlSql请求的响应，其操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| MMS | uint32 | 具体如何解析该参数，有系统管理服务器定义【可参考SSN -MDN-MUN这种方式】 |
| ClientSession | uint32 | ClientXmlSqlRequest. SessionId |
| Result | string | <Result>…</Result>语句  该消息会多次返回，通过扩展Result节点属性Count来控制总记录数。 |

XmlSqlNotice用于系统管理服务器给客户端的XmlSql通知(没有响应)，其操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete |

系统管理服务器将通知挂在该服务器上的所有客户端(用ROLE列表来过滤)。触发条件是：应用数据上报(参考3.5.2)、数据同步引发的变更(参考3.3.1)。同一个变化，可能会有多个客户端在监控，那么需要发出多条XmlSqlNotice

### 文件传输信令

主要用于客户端操作单板文件，在标准的文件传输信令上**前插**一个字段MDN和MUN路由信息字段（功能上可直接利用基础文件传输信令），如：

FtpOpenReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| MDN | Uint8 | 目标管理范围 |
| MUN | Uint8 | 目标单板系统 |
| RemoteFile | string | 远端文件名。 |
| Mode | uint32 | （1）0x01可写，0x02尾部读写，0x04清除[只有写的情况下清除选项才有效]。以上选项可以位或运算。  （2）如果是写文件，需要自动创建文件所在目录。  （3）能写就能读。 |
| ClientSession | uint32 | 本地会话标识，支持并发操作多个文件。 |
| Transfer | uint32 | （1）0=非传输；1=文件传输；为1时，Mode仅仅为0x00下载或0x01上传。  （2）使用无需响应的FtpWrite进行传输数据，传完之后发起FtpClose并进行校验(下载时，远端发起FtpClose；上传时本端发起FtpClose)。 |

其它消息请参考4.2.2.5.5，并依次类推的加MDN和MUN字段（操作码不变）。

该信令在单板系统管理模块一定需要做会话管理，

对于配置数据，如果第一阶段，想做的很简单，由客户端生成配置文件，传递到系统管理服务器，系统管理服务器下载到单板，单板通知应用，告知某文件发生变更，应用自己读取即可。

对于上述情况，系统管理服务器可以判断出该文件是程序文件还是配置文件，如果是配置文件，本地存储一份即可。

### 该接口上的其它建议：

#### 监控范围

监控在管理对象上的定义，在用户权限管理中约束。

对于监控哪些管理范围，还是全范围监控，在系统管理服务器通过配置实现。

#### 自发现管理

系统的拓扑变化主要包括：

1. 添加、删除网元或板卡
2. 添加、删除管理范围
3. 拓扑关系变化
4. 在现有拓扑关系中各个应用的运行状态变化。

这些系统变化，是可以让客户端自动发现的。只需要把‘发现’作为一种客户端的能力规约到用户权限中即可。也就把这类表格的‘监控’能力设置到相应客户端用户上，那么该客户端就能自动发现网络拓扑的变化。

#### 字典维护

应用的表格定义规约在TableInfo和FieldInfo两张系统字典表格中，客户端可通过操作这两张表，以创建、删除、修改表格定义（通过标准的Insert/Delete/Update语句进行），这样的话，应用可以不用手写表格定义的XML文件，通过客户端进行自动化创建、修改、删除表格定义，以减少错误率。

## E-IF2（单播接口）：

网络模型及基础通信协议请参考4.2。

### 客户端XmlSql操作信令

客户端请求Insert/Update/Delete是必须要发给单本模块的，对于Select却不一定，一般说来，在系统管理服务器就有了，但部分字段可能仅在应用中，那么就需要对部分字段的发起查询操作。

针对客户端XmlSql操作，我们定义两条序列化信令：BoardXmlSqlRequest，BoardXmlSqlResponse。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| BoardXmlSqlRequest | 13 | 单板发起的XmlSql请求 |
| BoardXmlSqlResponse | 14 | 应用回应的XmlSql响应 |

BoardXmlSqlRequest用于客户端发起XmlSql操作请求，其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ServerSession | uint32 | ServerXmlSqlRequest.ServerSession；单板不做会话管理。 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete/Select四种。 |

BoardXmlSqlResponse用于客户端发起XmlSql操作请求，其操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ServerSession | uint32 | ServerXmlSqlRequest.ServerSession；单板不做会话管理。 |
| Result | string | <Result>…</Result>语句  该消息会多次返回，通过扩展Result节点属性Count来控制总记录数。 |

单板不做该会话管理。

### 应用数据上报

这类情况主要针对故障、事件、状态的上报，落实到数据模型上讲，就是表格的曾删改操作，这类操作无需回应。

针对该类数据上报，扩展一个序列化操作：AppXmlSqlNotice。

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| AppXmlSqlNotice | 15 | 单板发起的数据变更通知 |

AppXmlSqlNotice操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| XmlSql | string | XML的SQL语句，支持Insert/Update/Delete三种。实际中往往只有Insert(故障、事件)和Update(状态)两种。 |

### 文件变更通知

对于配置数据，如果第一阶段，想做的很简单，由客户端生成配置文件，传递到系统管理服务器，系统管理服务器下载到单板，单板通知应用，告知某文件发生变更，应用自己读取即可。

当单板发现系统管理服务器下载了一份新的配置文件时，需要发起文件变更通知。

FileChangeNotice(文件变更通知)的操作码定义

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| BoardXmlSqlRequest | 16 | 单板发起的XmlSql请求 |

FileChangeNotice操作参数定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| FileName | string | 发生变更的文件名 |

# 模型设计

## 数据模型

本章建立的数据模型，是基于‘系统管理平台全面干涉应用数据’这一假设，包括数据解析和拼装两个过程。

为什么要进行‘全面应用数据干涉’呢？因为系统管理模块自身也需要一些配置数据，需要解析和拼装。如果能设计一个通用的数据模型，能表达应用业务的需求，那么一步到位是顺理成章，也不难的事情。

*如果第一阶段想做的很简单，也可以利用基础的通信模型和上层的协议接口完成（接口设计中将预留该支持）。*

### 表格化数据

二维表格化数据是关系型数据库的重要约定，某种意义上讲，任何数据都能用二维表格数据来表达。

数据模型表格化的另外一个重要目的是为了操作的模型化，对于表格，一般只有四类操作：插入、修改、删除、查询。这不仅仅是配置管理，还包括状态管理、故障事件管理以及应用控制功能都可以用表格化数据来表示。

按照这种模型，信令上就非常简单，总体上就只有四个业务操作信令，这对于信令的设计与平台的稳定都是非常利好的事情。最难模型化的业务是纷繁复杂的‘应用控制功能’，而这一块早在SNMP信令中得以体现和解决，SNMP定义的MIB数据规范中，使用‘control’类型的字段表示‘应用控制’的各类动作，其它字段表示该动作所需参数。通过这种方式也就把‘应用控制功能’映射到表格的‘Update’操作上了。在4.1.3章节中将详细说明表格模型。

#### 表格化.简单数据

实际应用中存在简单的非表格配置数据，如像INI文件配置一样：

[ShellClassInfo]

IconIndex=0

IconFile=C:\Program Files\Thunder Network\Thunder\Program\downloadfolder.ico

InfoTip=迅雷下载目录

这是‘迅雷下载’的配置文件，它的数据很简单，并不是二维的数据。对于这类数据也可以二维表格化，只需要约束表格ShellClassInfo的记录数有且只有一条即可，即单实例表格。

#### 表格化.树状数据

关系型数据库在表格化树状数据时，采用主从关系模型来表达【主键与外键】。当然关系型数据库的关系不只是主从关系。

树状数据用主从关系展开后，对于从表(或细表)还可以有更强的能力支持(如从表上实现多索引)，但是失去了主从操作的直观性。下面举例说明：

两张表格：‘成人’，‘孩子’。

用树状表示：

张三：

张老大

张老二

张老三

张老四

李四

李老大

李老二

李老三

如何获取‘张三’的‘老四’呢？“成人[‘张三’]. 孩子[3]”即可相当直观的获取。如果是纯关系型，一般会用复杂的SQL来表示。

这里有一个问题，如果’王五’是’张三’的妻子呢？如何表达，‘张老二’同时也是’王五’的孩子？再如’张三’有两个妻子呢？换句话说，树状表达在面临复杂关系时，是很难的。

**本文档所提供的设计是基于树状关系来设计的，且没有完全展开为纯关系型模型。为何呢？**

1. **传统网管标准，包括SNMP/TR069/LDAP所支持的数据模型也只是树状模型(其中SNMP是全展开，TR069和LDAP都直接用树表达)。**
2. **作为管理型数据确实存在树状特征，而纯关系型在表达树状模型上，不够直观。**

**即便有更强烈的关系性需求，也可通过上层应用自行关联。**

### XML模型抉择

在数据通信和数据的文件存储方面，将全面采用XML格式。然而XML从诞生到现在，已非常庞大，也有很多分支，我们该选择什么样的XML数据模型呢？

首先我们从基本的XML语义要素上分析，如下：

1. 头部分：
   1. 非实际数据内容。
   2. 主要描述一些版本，规范、语言等信息。
2. 节点
   1. 是实际数据内容。
   2. 节点(可称之为对象)可包含属性、元数据、子节点三个部分，如：

<person id="person\_id\_01">

<stuff\_id>1</stuff\_id>

<address size="32">shenzhen</address>

<age>25</age>

<gender>0</gender>

<url>www.google.com.hk</url>

</person>

其中id和size都是属性

stuff\_id、address、age、gender、url是节点person的子节点。

1、shenzhen、25、0、[www.google.com.hk](http://www.google.com.hk)是相应节点的元数据。

在描述数据时，可以选择属性或元数据，但没有强制的约定到底用哪个。一般说来，属性用于表达简单的、可预见的数据，元数据用以表达复杂的或不可预见的数据。

标准化的结构化数据描述有XPoiner、XPATH、XSD、XML-DB等。这里大部分都基于元数据的描述方法（因为他们企图描述通用的数据模型）。

XSD比较易用，可让应用层面关注较小，也正是因为如此，有时反而不好用，例如凡是结构化数据的翻译工作，都需要自动转化为一个函数实现，那么当我们需要新定义一张应用的配置表格时，是需要新作代码的，这对平台来讲，新业务的开发是很不利的。所以本文档不选择XSD作为描述规范。

另外XSD也基于XPATH/XPointer进行数据访问，XPATH/XPointer的访问规约是通过下标进行。如，“成人[1]. 孩子[3]”，不能基于条件查询，定位工作由应用自行计算好，这是有问题的，我们在TR069的实践中发现了一个大问题，就是当两个客户端同时访问一张表格时，如果其中一个客户端发起了对表格记录的删除，会导致另外一个客户端的访问错乱，当时的解决方案是同时只允许一个客户端对同一个表格进行访问。这种表格加锁对系统带来的设计难度是很大的，现实运行中也是容易出问题的地方。

XML-DB没有一个标准的语法规范，也没有一个标准的独立实现，都是和自己的数据库系统绑定的，不同数据库厂商略有不同，大体上都是利用元数据部分传递SQL语句的核心部分，利用属性提供XPATH信息。

**我们的数据模型其实并不需要很复杂，本文档的设计中仅仅利用了节点和属性两个元素就能满足了，也鉴于上述分析，决定自行设计XML的数据模型，而不直接采用XSD等规范。**

### 表格模型描述

本章节所描述的表结构信息，只是一个模型，并不存在真实的XML样式。

本章所描述的表格模型，企图能涵盖：配置数据、状态数据、故障事件数据、以及应用控制等。

#### TableInfo

表格的最顶端定义，由XML节点标记TableInfo括起来，即：

<TableInfo >

<!-- Field Info -->

<!-- Index Info -->

<!-- Detail Table Info -->

</TableInfo>

TableInfo节点的属性定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Name | string | M | 描述表名，需要符合文件名规范，可不分大小写（对应的文件名全小写）。 |
| Single | bool | O | (1)允许的取值范围为：true, false, yes, no,1,0  (2)用以表示该表格是单实例表还是多实例表。  (3)缺省为false |
| MaxInstance | uint32 | O | 当多实例表时，用以指定最大记录数。缺省不做最大记录数限制。多实例表最大记录数为0时，表示不存储，比如事件表【可通过监控接口通知客户端】。 |
| OverFlowPolicy | enum | O | 当多实例表有最大记录数限制时，该表数据满额后的Insert策略定义，现预定义如下【可扩展】：  0=失败；  1=删除最先插入的记录  2=删除根据时间字段显示的最老记录 |
| TimeField | string | O | 当OverFlowPolicy为2时，指定的时间字段名。 |
| Dynamic | enum | O | (1)取值范围：fixed, static, dynamic, 仅仅限制客户端是否允许动态增删记录。  (a)固定表格(fixed)：这类表格需要必须要有出厂数据，在现场是不允许进行增删记录。（系统管理服务器需要参考该约束）。  (b)静态表格(static)：在相关应用或服务模块运行期间，不允许增删。（单板系统管理模块需要参考该约束）。  (c)动态表格(dynamic)：在相关应用或服务模块运行期间，允许增删。（单板系统管理模块需要参考该约束）。  (2)缺省为dynamic。  **如果实现难度大，可考虑分阶段实现。** |
| DbName | string | O | 1. 数据库表格的数据库名，如果为空表示非数据库表格。 2. 数据库表格可支持复杂查询。 3. 数据库表格主要针对用户操作日志、事件、故障等。 |
| SDN | int | M | (1)同步范围编号，-1表示该表不参与同步，是本地表格。  (2)缺省为0。 |
| MD5 | uint8[16] | O | 对于数据库表格就不需要了。  记录该表格对应文件的MD5，用于4.3.4.1流程。  MD5字段在加载表格文件时计算，以及刷新数据时重新计算。当然该字段不一定存储在该表格对应的字典文件中，而是需要单独维护下载文件列表 |

1. 细表是主表记录的属性扩展，多实例表必须有索引，如果应用确实没有可索引的字段，也需要添加辅助的索引字段。以此支持树状的数据表达。
2. 数据库表格不能是细表，也不支持它的细表。数据库中的表格采用单独的SQL脚本创建。
3. SDN，请参考数据同步一章节的设计。
4. MD5属于逻辑属性，不一定需要维护在该表格中。

#### FieldInfo

<TableInfo> … </TableInfo >内可包含多个字段描述，每个字段由<FieldInfo/>表示。

FieldInfo节点的属性定义为：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Name | string | M | 描述字段名，可不分大小写。 |
| Type | enum | M | string, 普通文本  name，定义一种标识符规范的文本数据  filename，定义一种满足文件名规范的文本数据  int8,int16, int32, int64,uint8,uint16,uint32,uint64，整型  bool, 取值范围为'1'、'0'、'true'、'false'、'yes'、'no'  enum，整型的枚举类型  datetime UTC时间格式的字符串，**如果第一阶段不好实现，可不考虑该类型**。  base64, base64编码串  binary, 二进制数据串 |
| case | bool | O | (1)仅针对文本型数据有效（string & name）  (2)True:大小写敏感；false：大小写不敏感。  (3)缺省为true。 |
| Range | string | O | (1)缺省不做范围限制。  (2)由{}括起来的多个范围组合，范围之间用逗号分隔  (3)范围可由冒号分隔，如:  (a)a:b，表示最小为a，最大为b  (b)a:，表示最小为a，无最大限制  (c) :b，表示最大为b，无最小限制  (d) c，表示为单值c，即c:c的简写，enum的枚举项只能是单值  (4)string, base64,binary的范围表示长度范围  (5)整型的范围表示数值的取值范围。  (6)datetime表示时间范围 |
| NotNull | bool | O | 1. 表示该字段是否允许为空。 2. 缺省为false； |
| Default | string | O | 1. 指定缺省值时，一定不允许为空(忽略NotNull)。 2. 当插入新纪录没有指定该字段值时，需要用默认值。 |
| Dynamic | enum | O | 1. 取值范围：read-only, static, dynamic，仅仅限制客户端是否允许动态修改。 2. 只读字段(read-only)，表示该字段是只读字段，不可以修改。（系统管理服务器需要参考该约束） 3. 静态字段(static)，在相关应用或服务模块运行期间，不允许修改。（单板系统管理模块需要参考该约束） 4. 动态字段(dynamic)，表示该字段的修改可实时生效。（单板系统管理模块需要参考该约束） 5. 缺省为dynamic，**如果实现难度大，可考虑分阶段实现。** |
| Storage | enum | O | 1. 取值范围为：persistent, temporary, control 2. 持久性数据(persistent)，需要存储于文件或数据库中，系统重启能恢复的数据。 3. 临时性数据(temporary)，仅存在于内存中，系统重启后就丢失了，可重新获取，往往状态性数据就是这样的。 4. 控制性数据(control)，该字段必须允许为空，连内存中都不存在，用以控制应用操作，如单板的上下电等。 5. 缺省为persistent |
| Location | enum | O | 1. 取值范围：    1. system(persistent,temporary,control)    2. application(temporary,control) 2. 系统侧数据(system)，表示数据以系统管理平台为准，并存储于系统管理平台中。应用可以上报该项临时数据(状态)或持久数据(如故障事件)。 3. 应用侧(application)，表示数据存储于应用应用侧，通常状态数据就存储于应用侧。 4. 缺省为system |
| Primary | bool | O | 1. 用以指示主键字段，一张表可以含由多个主键字段构成唯一主键索引。 2. 主键字段是只读的，并不能为空【忽略Dynamic和NotNull】。对于主键的修改由客户端发起先删后插的流程完成。 3. 缺省为false |
| Sync | bool | O | 1. 指示该字段变更，是否需要同步到对端。 2. 缺省为真，表示需要同步。 |

1. 对于实际字段的取值范围，可能还会有更进一步的语义要求或上下文要求，如：
   1. 满足某种格式的字符串，甚至这种格式可定义，像正则表达式一样。
   2. 根据其他数据(可能来自其它表格)的组合、或部分、或变换等。

对这类要求，约定由客户端实现，在系统管理平台不做约束。

1. Sync，请参考数据同步一章节的设计。
2. 关系型数据库系统，仅允许定义一个主键字段，而网管系统是可以定义多个主键字段的。

### 系统目录的规划

该章节直接影响到应用管理、升级等业务流程和信令接口。

单板系统的大体启动流程：

1. 首先要连接到系统管理服务其中。
2. 下载一个需要下载的文件列表。
3. 本地进行MD5码校验。
4. 对MD5码校验不同的文件发起下载流程。
5. 启动应用。

这里要配置文件下载，但为了和升级管理等不冲突，最好和程序文件不要放在相同目录，以防软件升级时把配置文件覆盖掉。

应用系统在运行期间，可能还会动态生成并维护一些数据文件(非网管数据)，对于这类数据也最好不要和程序文件及网管数据文件冲突。

所以为了方便处理，本设计约定：单板系统分三大目录进行存放各类文件：

1. AppHome、用于存放应用软件(包括单板系统管理模块程序文件)
2. NmsHome、用于存放网管配置文件
3. RunHome、用于存放应用自身管理的文件（包括单板系统管理模块自身所管理的文件）,例如4.1.7.4定义的操作日志文件。

这几个顶级目录可通过环境变量指示。

### 统一编码

#### 管理范围编码

使用MDN(Manage Domain Number)管理范围编号为系统管理服务器进行逻辑编号。约定0xFF为无效值。

一个系统管理范围允许多个系统管理服务器，但是都拥有相同的系统管理编号，即从逻辑上只有一个系统管理服务器。

*在实际项目或产品中可用框号或框类型进行编码，因为某类单板可能只能插在某类机框上。*

#### 单板系统编码

使用MUN(Uni-board System Number) 单板系统编号为单板系统管理模块进行逻辑编号。一个系统管理范围可管理多个单板系统管理模块。约定0xFF为无效值。

单板系统管理模块自身也是一个应用，也需要按照应用系统编号法则进行进一步编号【参考4.1.5.3】。

*在实际项目或产品中可用槽位号或单板类型进行编码，为什么要涉及到单板类型呢？因为可能某类应用只能运行在特定硬件单板上。*

#### 应用系统编码

使用SSN(Sub System Number) 子系统编号为某类应用进行逻辑编号。同一单板可管理多类应用，但约定不能同时运行某类应用的多个实例。约定0xFF为无效值。

某类应用可以同时运行在不同的单板上【可属于相同或不同的管理范围】。

SSN是一种系统资源，需要集中统一管理，每开发一个新的应用，都需要申请一个SSN，**预定义SMS的SSN为0，SMU的SSN为1**，其它应用后续定义。

#### 管理对象编码

我们用OMI(Object Managed Identity) 管理对象标识为管理对象进行编码。根据数据模型的设计，我们可用表名来代表管理对象标识。并且根据表的树状特征，那么OMI也需要能表达层次关系。

在考虑到实际的网络部署，我们可以MDN-MUN-SSN-OMI来表示一个实际部署的管理对象，并用MMS(MDN-MUN-SSN)来表示管理对象的位置编码。

对于客户端而言MMS可以由系统管理服务器自行定义，对对于系统管理服务器和单板系统管理模块必须是明确的MDN-MUN-SSN语义。

### XML-SQL

本节定义基于XML的结构化查询语言，需要说明的是，国际标准中虽然有XML-SQL提法，但没有统一的规范，不同数据系统厂商，提供的实现都略有不同。

这里根据系统管理自身的特征及要求，定义自己的规范，下表预定义几个概念：

|  |  |
| --- | --- |
| 概念 | 说明 |
| 语句 | 定义了Insert、Update、Delete、Select、Result等5种语句 |
| 子语句 | 5种语句可以嵌套自身的语句称为子语句，这是为了表达树状的数据模型。 |
| 子句 | 定义了Set、Where、Record、Field四种子句。 |
| 叶子语句 | 不含子语句的语句，以及最底层的子语句称为叶子语句。 |
| 分支语句 | 把含有子语句的语句称为分支语句。 |
| 根语句 | 不含子语句的语句，以及最顶层的分支语句称为根语句。 |

#### Insert语句

<Insert>

<!--

Field子句，参考4.1.6.6.4

如果该子句在Insert分支语句中，仅包含所有主键字段，用于定位细表。

如果该子句在Insert叶子语句中，可以包含各种字段，用于构建新记录。

-->

<!--

可选Insert子语句，如果操作的是细表则需要包含Insert子语句，结构上和Insert语句类似。

-->

</Insert>

<Insert>节点属性定义

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Table | string | M | 描述表名， |

样例：

<insert table=” person”>

<field name=” stuff” value=” 1”/>

<field name=” address” value=” shenzhen”/>

<field name=” age” value=” 25”/>

<field name=” gender” value=” 0”/>

<field name=” url” value=” www.google.com.hk”/>

</insert>

#### Update语句

<Update>

<! --

Set 子句，请参考4.1.6.6.1

该语句仅存在于Update叶子语句，用以表示需要修改的字段。

-->

<!--

Where 子句，请参考4.1.6.6.2

该子句仅存在于Update叶子语句。用以定位修改记录。

-->

<!--

Field子句，参考4.1.6.6.4

如果该子句仅存在于Update分支语句中，仅包含所有主键字段，用于定位细表。

-->

<!--

可选Update子语句，如果操作的是细表则需要包含Update子语句，结构上和Update语句类似。

-->

</Update>

< Update >节点属性定义

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Table | string | M | 描述表名， |

样例：

<update table=” person”>

<set>

<field name=” address” value=” shanghai”/>

<field name=” age” value=” 30”/>

</set>

<where>

<field name=” stuff” value=” 1”/>

</where>

</update >

#### Delete语句

<Delete>

<!--

Where 子句，请参考4.1.6.6.2

该子句仅存在于Delete叶子语句。用以定位修改记录。

-->

<!--

Field子句，参考4.1.6.6.4

如果该子句仅存在于Delete分支语句中，仅包含所有主键字段，用于定位细表。

-->

<!--

可选**Delete**子语句，如果操作的是细表则需要包含**Delete**子语句，结构上和**Delete**语句类似。

-->

</Delete >

< Delete >节点属性定义

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Table | string | M | 描述表名， |

样例：

<delete table=” person”>

<where>

<field name=” stuff” value=”2”/>

</where>

</delete >

#### Select语句

<Select>

<!--

可选Where 子句，请参考4.1.6.6.2

该子句仅存在于**Select**叶子语句。用以定位修改记录。

如果没有该子句，表示查询所有记录。

-->

<!--

Field子句，参考4.1.6.6.4

如果该子句仅存在于**Select**分支语句中，仅包含所有主键字段，用于定位细表。

-->

<!--

可选**Select**子语句，如果操作的是细表则需要包含**Select**子语句，结构上和**Select**语句类似。

-->

</Select>

< Select >节点属性定义

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Table | string | M | 描述表名， |
| Filter | string | C | 仅存在于Select叶子语句中。用于表示查询的字段:  (1)\*表示所有字段；（缺省值）  (2)多个字段用逗号分隔。 |

样例：

<select table=”person” filter=”suff,age, gender”>

<where>

<field name=” age”, value=”50” compare=”ge”>

</where>

</select

#### Result语句

该语句用于表示Insert、Update、Delete、Select语句执行的结果。

<Result>

<!—

可选Record 子句 参考4.1.6.6.3

用于表示Select返回的结果集

-->

</Result>

Result属性定位

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Code | uint32 | M | 0=操作成功；1=系统正忙；2=其它错误；3=认证失败；4=表不存在；5=定位失败；6=不允许操作；7=无效数据；8=插入冲突；9=容量满额。  Insert：0~9；Update：0~7；Delete：0~7；Select：0~7 |
| Count | uint32 | C | 可选属性，用于表示Select查询返回结果集的记录总数。  该语句嵌入的具体消息可分多次返回。每次返回可按单条或多条记录返回。 |

Select的Result样例

<result code=”0” count=”10”>

<record>

<field name=” stuff” value=” 100”/>

<field name=” age” value=” 50”/>

<field name=” gender” value=” 0”/>

</record>

</result>

<result code=”0” count=”10”>

<record>

<field name=” stuff” value=” 101”/>

<field name=” age” value=” 52”/>

<field name=” gender” value=”3”/>

</record>

</result>

#### 子句

##### <Set>…</Set>子句

<Set>

<!--

Field子句，参考4.1.6.6.4

用以表示需要修改的字段。

-->

</Set>

该语句用于Update语句中。

##### <Where>…</Where>子句

<Where >

<!--

Field子句，参考4.1.6.6.4

在Update&Delete语句中，仅包含主键字段值，以限制仅作唯一性定位。

在Select语中可以携带任何字段，以支持多条记录的查询。

-->

</Where >

如果需要批量修改或删除数据，由客户端自行设计。

##### <Record>…</Record >子句

<Record>

<!—

Field子句，参考4.1.6.6.4

这里不包含主表的主键字段

-->

</Record >

##### <Field/>子句

该子句节点没有子节点，其属性定义为：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Name | string | M | 描述字段名。 |
| Value | string | M | 描述字段值。 |
| Compare | string | O | 1. 仅用于Select的Where子句 2. lt，le，eq，ge，gt 3. 缺省为eq |

如果用在Result语句中，不包含主表的主键字段。

### 数据的文件存储

#### 数据存储总体方案

##### 公共配置的存储

一个管理范围内需要能管理自治，那么跨管理范围的公共配置理应在不同管理范围内独立存储。但是同一管理范围内呢？

1. 不同应用的公共配置，是分多份还是单份？
2. 同一应用的不同运行实例的配置，是分多份还是单份？（尤其是公共配置，因为同一应用，可能大部分配置是相同的，只有少部分配置不同）
3. 不同应用表格重名，又如何存储？
4. 多份数据。这种方案对于通信接口比较简单，但对于表格在系统管理服务器中如何组织，就值得仔细思量的了。另外，相同应用的不同实例，可能大部分配置是相同的，只有少部分配置是不同的？是让用户做多份配置，还是让客户端来屏蔽。
5. 如果是单份，就得考虑应用实例的编号如何在表格中体现？在通信接口层面如何传递。相同应用的不同实例的公共配置，对于系统管理服务器的协议和流程都是有影响的，需要向多个应用实例上发送。

这里为了简化平台的管理方案，决定采用多份存储：即按如下的路径进行分级存储。$NmsHome/MUN/SSN/，其中MUN和SSN请参考4.1.5章节。

##### 整体存储还是按表独立存储

系统的所有数据组成一颗大树，这颗树上有很多分支，众多分支中有些是根节点，有些是分支节点，有些是叶子节点。是按根为单位存储还是按表为单位存储呢。

###### 以根为单位存储（？）

如果以根为单位存储，那么该根及其分支和叶子数据均存在于一个文件中，那么文件名可以是’根表表名.xml’。

这样做带来的问题是，如果某个从表的变化，会导致整个文件的重写，这对文件的IO产生重大影响。

###### 以表为单位存储（？）

不同的表存储于不同文件，这样某个从表的变更，只会影响单个文件的变更，可降低对文件IO的影响。

但是不同分支下的表名如果相同，如何标识？可以有两种做法：

(1)文件分目录存，每一个表名是一个目录名，并根据主从关系构成一致的目录树，每个目录下都用统一的文件名标识数据，如Table.xml。

(2)所有数据文件放在同一目录下，不过在文件名上需要有其主表的名字信息，可用句点号分隔，与(1)同理，文件名也能表达出与系统一致的主从关系。

但是这两种做法都有相同的问题，就是整个系统对文件命名是有总长度限制的，如Windows XP下只有260个字符，那么层次太深时，如何分表格存储呢？

另外一种方案就是强制系统所有表名均不相同，就像纯关系型数据库系统一样，那么每个文件就可以直接使用’表名.xml’了。

*我们是在文件名长度上做限制呢？还是约定为表名不同？*

###### 融合方案

以表存储，可降低对文件IO的影响，但有文件名长度的问题。以根存储【至少分应用或分板】，可以避开文件名长度的问题，但是文件IO量会加大。

对于配置型数据，其实变更量是不大的，变更量大的可考虑存储于数据库中。那么事实上是可以考虑按根存储的，可通过4.1.7.4章节所述的操作日志的方式，定期刷新数据到文件，那么可以降低对文件IO的影响。如果要考虑做成一种通用的数据管理模型，在后续其它应用也能用得上，那么两种方式都是可以考虑的【通过配置或应用指定的方式识别到底是那种存储方案】。

#### XML文件格式

##### <Table> … </Table>

特定表格的数据用<Table> … </Table>括起来，即：

<Table>

<!-- Record -->

</Table>

<Table>节点的属性定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Name | string | M | 描述表名，需要符合文件名规范，可不分大小写（对应的文件名全小写）。 |

1. 只要某张表格的数据具有persistent类型的字段，该表格就需要文件存储。
2. 如果主表不满足文件存储要求，细表也不会被存储【即便满足文件存储要求】。
3. 文件名为” $NmsHome\$AppName\config\OMI.xml”【OMI参考4.1.5】

##### <Record> … </Record >

表格中的记录由<Record> … </Record >括起来，一张表格可以包含多个<Record>子节点，即：

<Record>

<!—Field Data -->

<!—Detail Table Data -->

</Record >

因为按表分别存储，为了正确加载数据，从表中也要记录主表的主键字段，也从根往叶子方向嵌套存储。

<Record>节点的属性定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Time | datetime | M | 最后一次修改时间 |

该属性为后续的一致性检查做铺垫，可暂不考虑实现。

##### <Field/>

< Field>节点的属性定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 数据类型 | 选项 | 属性说明 |
| Name | string | M | 描述字段名。 |
| Value | string | M | 描述字段值。 |

1. 只有persistent类型的字段，才会被存储。
2. 字段为空时，不存储。

#### 内建数据字典（~O~鸡与蛋的故事~O~）

$AppHome\$AppName\Config\TableInfo.xml

$AppHome \$AppName\Config\FieldInfo.xml

这两个文件放在AppHome目录下的原因是这两个文件，是运行期不变的，除非版本升级。

这两张表归系统管理服务器和单板系统管理模块使用。

这里约定所有应用的文件部署方式（可商量，但必须作为一种默认不可配置的规则）。

其中AppName表示应用名字，所有应用都放在同一个目录中。把配置字典即数据放在应用自身的目录下，数据字典的修改，可随应用发布，系统管理平台可以不做任何的发布工作。

TableInfo&FieldInfo定义该应用的所有表的数据字典，而TableInfo&FieldInfo本身也是表格，但TableInfo&FieldInfo自身的数据字典采用内建的方式。

数据字典系统表TableInfo的内建定义约定如下**（没有给出的属性均采用默认属性值）**：

<TableInfo Name=” TableInfo” >

<FieldInfo Name=”TableName” type=”name” case=”false” Index =”true”/>

<FieldInfo Name=”Single” type=”bool”/>

<FieldInfo Name=”MaxInstance” type=”uint” />

<FieldInfo Name=”Dynamic” type=”enum” enum=”{fixed=0, static=1, dynamic=2}”/>

<FieldInfo Name=”DbName” type=”string” />

<FieldInfo Name=” BoardStroage” type=”bool” />

<FieldInfo Name=”SDN” type=”int” range=”{-1:65535}” default=”0”/>

<TableInfo Name=”FieldInfo” >

<FieldInfo Name=”FieldName” type=”name” case=”false” NotNull=”true” Index =”true”/>

<FieldInfo Name=”Type” type=”string” NotNull=”true”

enum=”{string,name,int,uint,long,ulong, bool, datetime,base64,binary}”/>

<FieldInfo Name=”case” type=”bool”/>

<FieldInfo Name=”range” type=”string”/>

<FieldInfo Name=”enum” type=”string”/>

<FieldInfo Name=”notnull” type=”bool”/>

<FieldInfo Name=”Default” type=”string”/>

<FieldInfo Name=”Dynamic” type=”enum” enum=”{read-only=0, static=1,dynamic=2 }” />

<FieldInfo Name=”location” type=”enum” enum=”{system=0, application=1}” />

<FieldInfo Name=”storage” type=”enum” enum=”{persistent=0, temporary=1, control=2}” />

<FieldInfo Name=”Primary” type=”bool”/>

<FieldInfo Name=”Sync” type=”bool” default=”true”/>

</TableInfo>

</TableInfo>

系统启动时可根据TableInfo&FieldInfo，创建应用表格。

**该数据字典的维护可通过系统管理客户端发起通用的表格操作来进行，系统管理平台处于特定的运行模式才支持该表的维护，也希望系统管理客户端也在特定模式下才支持对该表的维护。**

#### 操作日志

针对要做本地存储的表格，其“插入、删除、更新”操作，涉及到文件更新的操作时，需要写到操作日志中【包括用户操作日志和数据同步产生的日志】，当系统重启时，首先加载XmlSqlLog.xml，再加载操作日志。

系统定期刷新所有数据到文件中，表格对象需要标记该表是否变化，刷新后清空该操作日志文件。

该操作日志文件，是操作的XML-SQL文本文件（参考4.1.6）。其大小控制可通过配置决定。当写满后，也要发起立即刷新流程。

系统启动完后，也需要发起立即刷新流程。

### 数据的数据库存储

详细设计文档，做详细设计说明，这里只做要求，提供一个通用的API接口，用于数据库操作，包括四个数据操作（Insert, Delete, Update, Select【需要支持多批返回】）和两个字典维护操作Create， Drop等。Create用于创建数据库、表格、索引、视图等，需要考虑对象已存在，并自动进行修改的能力支持。

下层对接的数据库，可先以SQLITE为例。

## 通信模型

### 组网模型

#### 子网划分



这里以系统管理服务器为中心，把涉及到与之通信的网络实体的集合定义为子网。

根据管理范围SMD(System Manage Domain)的定义，我们划分两个子网，管理范围内的子网叫内子网，跨管理范围的子网叫外子网。外子网包含所有内子网的系统管理服务器节点。

单板系统(Uni-System)和客户端均仅具有一个子网，系统管理服务器具有两个子网。

子网划分的目的：

(A).客户端的安全访问和正确寻址（参考3.6章节）。

(B).减少数据同步量（参考3.3.1章节）。

(C).访问的简单性：

内子网的系统管理服务器对单板系统和客户端而言是一个逻辑实体，实际上会落在某一个系统管理服务器上，我们把这个服务器称为**活动系统管理服务器。**

由于有多个管理范围，上述机制仅仅降低了单板系统的接入难度，对于客户端而言并没有，所以我们再进一步在外子网上再竞选出一个**全局系统管理服务器，**作为客户端的统一接入实体。为了降低客户端对**全局系统管理服务器**的访问负荷，客户端仅从**全局系统管理服务器**获取当前的**活动系统管理服务器**列表**，**客户端随机选择一个进行访问（具体细节请参考3.6章节）。

外子网中，也不是所有系统管理服务器都能与客户端连接。换句话说外子网中还可以包含一个客户端的接入子网。这里不去划分单独的子网，我们将在虚拟IP设计章节屏蔽这个问题。

#### 管理IP划分

##### 客户端的管理IP

可用主机IP地址，即C-PIP。

##### 单板系统的管理IP

也可用主机IP地址，即B-PIP。

##### 系统管理服务器的管理IP

由于系统管理服务器包含两个子网，这两个子网可能在物理网络上是相通的，也可能是隔离的(如双网卡隔离)。这里进一步区分这两种情况。

###### 内外子网在同一网络平面



每个系统管理服务器有一个物理IP，即S-PIP。

为全局系统管理服务器提供一个VIP，即G-VIP，供客户端首次连接使用。请参考3.6 E-IF1接口。

为每个活动的系统管理服务器提供一个VIP，即S-VIP，用于系统管理服务器与单板系统及客户端的互联，请参考3.2 I-IF1接口和3.7 E-IF2接口。

所有的S-VIP、G-VIP、S-PIP均要求不同。

###### 内外子网在不同网络平面



为了安全，可用双网卡实现物理隔离，形成不同网络平面。那么系统管理服务器就会有两个物理IP，即S-I-PIP和S-E-PIP，其中S-I-PIP用于对内提供信令承载服务，S-E-PIP用于对外提供信令承载服务。

S-I-PIP用于管理范围内的系统管理服务器互联；S-E-PIP用于管理范围之间的系统管理服务器互联。

为全局系统管理服务器提供一个VIP，即G-VIP，供客户端首次连接使用。请参考3.6 E-IF1接口。G-VIP与S-E-PIP绑定在相同网卡设备上。

为活动的系统管理服务器提供一个内部VIP，即S-I-VIP，用于系统管理服务器与单板系统的互联，该S-I-VIP与S-I-PIP绑定在相同网卡设备上。

为活动的系统管理服务器提供一个外部VIP，即S-E-VIP，用于系统管理服务器与客户端的互联，该S-E-VIP与S-E-PIP绑定在相同网卡设备上。

如果内子网和外子网的网络是隔离的(双网卡可实现)，那么系统管理服务器就会有两个IP，一个IP用于系统管理服务器与单板系统之间的互联，另一个IP用于系统管理服务器之间互联以及系统管理服务器与客户端之间的互联。

所有的S-E-VIP、G-VIP、S-E-PIP不能相同，同一范围内的S-I-VIP、S-I-EIP不能相同。所有的S-I-PIP可以相同(建议)，所有的S-I-VIP可以相同(建议)。同一管理范围之内的单板物理IP不能相同。

###### 融合方案

（1）需要一个配置选项，指示网络组网方案，以区分是同一网络平面还是不同的网络平面。

（2）按内外子网在不同网络平面作为基础的设计方案，应对同一网络平面情况是，可按照下述方法解决：S-I-PIP取用S-E-PIP，S-I-VIP取用S-E-VIP相同。

（3）这些IP地址，不能通过网管配置，可通过其它途径(如DHCP)方式或按照某种规则计算获取。

##### IP绑定建议

针对VxWorks系统，一般系统启动时，是没有IP地址的，都是在启动过程中通过配置绑定的。

对于Linux系统，往往记住了应用绑定的IP地址，这对于IP地址的管理反而不方便。当在从网管上修改某个IP地址时，单板系统在修改前，发生异常而导致板卡重启，重新启动后，网管的配置已经修改，而单板应用不知道以前的IP地址，无法清除以前的IP地址。最小的问题，会导致单板IP地址垃圾，大问题是单板IP地址冲突。

可在Linux启动时做一个内存盘，把关于所有无需保存的文件放在该盘上【包括记录该IP地址的文件】，那么系统重启时所有IP地址都是空的，可用应用进行重新绑定。另外一种做法是：

1. 获取IP地址列表。
2. 清除系统现有的所有IP地址。
3. 重新绑定IP地址。

总之，单板出厂时需要能有一个默认IP，以便接入系统，获取真实的获取IP地址列表。默认IP可通过DHCP服务获取一个内网用的默认IP。

### 可靠通信

#### 网络承载协议选择

有很多协议可供选择，如TCP、UDP、SCTP、IPX、SPX等网络协议。为了降低系统要求，建议在最普通的TCP和UDP之间选择【其它协议需要特殊的软件包支持】。

选择TCP的好处在于自动实现传输的可靠性，不好的地方在于需要管理连接，针对每个连接需要独立的接收线程，比较浪费资源，当然也可以采用异步机制，但处理过程比较复杂。

数据同步这块，肯定需要选择多播UDP，TCP是不行的（因为需要两两建立连接，这个模型很丑陋），选择单播UDP的话，需要在不同UDP上发送数据，复杂度增大。既然选择了多播UDP，就得考虑消息的可靠性传输。

那么在一对一的通信中，如何选择呢？如果在多播的可靠性传输设计上兼容单播UDP的可靠性传输，那么单播UDP也是一个不错的选择。

**所以最终决策：全面采用UDP协议。**另外在一对一通信中，可以很容易设计出类似于SCTP的偶联特征，即：‘A连B’和‘B连A’完全等价。TCP是做不到这点的，TCP必须是两条双向链路，SCTP中是一条双向链路。

#### 消息序列化设计

##### 序列化模型

###### 组播序列化模型



###### 单播序列化模型



##### 序列化队列

对网络上接收到的消息按照接收的时序进行排队，是一个循环队列。上图其中0表示该序号对应的消息丢失，1表示该序号对应的消息未丢失，2表示再也取不回来消息了(将忽略处理)。

消息入队列机制，如果是新消息，放置队列尾端t，t向右移动。如果是旧消息，则填入相应位置（由滑窗控制识别消息的新旧）。

消息出队列机制，用上图说明，当头端h空消息处填充好后，立即启动消息出队列流程，弹出一个处理一个，h向右移动，知道h到达t或空消息位置。

消息的入队列和出队列由同一线程完成。

序列化消息队列节点记录消息的内容和发端节点编号。

##### 接收滑窗

对某个节点的接收消息进行按消息序号排队，滑窗节点项记录消息属性：(1)是否为空，(2)对应在序列化队列的位置。

接收端滑窗记录头/尾端的消息序号。接收到某条消息，需要判断该消息是否过期【丢包时间太长，启动过异常处理机制】，还需要判断某条消息是否为新消息，若是新消息，与尾端消息相差多少个丢包消息。

当序列化队列头端右移时，根据该消息发端节点编号，对相应接收端滑窗进行头端右移。

###### 滑窗重要参数

nSize，表示滑窗大小。

nHead，表示滑窗的头部消息序号。取值范围：[ 0，nSize-1 ]。

nTail，表示滑窗的尾部消息序号(下一个应收的消息)。取值范围：[ 0，nSize-1 ]。

bFirst，表示系统没有放入过消息。

滑窗存储的消息范围为：[ nHead，nTail )。

初始：nHead = 0；nTail = 0；bFirst=true；

收到第一个序列化消息时，会执行下述逻辑。

if(bFirst)

{

nHead = nSeq;

nTail = nSeq;

bFirst = false;

}

###### 消息存放位置算法

nDistance = nSeq % nSize

其中nSeq表示当前消息的序号。

nDistance表示应该存储在滑窗中的位置，如果用数组实现滑窗oWindow，那么oWindow [nDistance]就是代表消息存放所在的内存空间。

###### 消息过期算法

nDistance1 = nSeq - nHead;

if(nDistance1 >= nSize)

{

if(nDistance1 <=( 0xFFFFFFFF - nSize + 1))

{

//可能是系统重启或长时间断链导致。

//滑窗异常处理，继续后续流程。

}

else

{

//消息过期，可丢弃当前消息，中断当前流程。

}

}

###### 新消息与旧消息的判断

nDistance2 = nTail - nHead;

if(nDistance1 >= nDistance2)

{

//表示nSeq对应的消息是新消息

//nSeq - nHead表示应该插入的空消息数。

//如果空消息插入序列化队列失败(满了)，需要启动滑窗异常处理。

}

else

{

//表示nSeq对应的消息是旧消息

//nSeq% nSize表示消息应该存于何处。

}

###### 滑窗异常处理

while(nHead!=nTail)

{

nDistance = nHead % nSize

if(oWindow[nDistance]是空消息)

{

//那么在序列化队列的相应位置也应该是空消息

//应该认为该空消息再也取不回来了，设置为2，让系统继续能处理。

}

++ nHead;

}

bFirst = true;//很重要，表示下次收到消息，将重新开始。

除了消息序号在可承受范围内过期需要进行滑窗异常处理外，收到注销消息，也需要进行滑窗异常处理。

##### 保持队列

当开启序列化服务后，允许接收序列化消息，但是由于某种原因，致使上层应用模块还不能真正成为服务状态时，序列化消息出队列后，需要放入保持队列中，当上层应用模块真正成为服务状态时，才开始序列化消息的处理。

直到保持队列为空时，序列化消息即转入即时处理。当然该队列是可选的。

##### 发送队列

###### 组播发送队列

发送端队列是一个循环队列，队列满时循环覆盖。

###### 单播发送队列

发送队列中的消息分两类：（1）群发消息，（2）非群发消息。使用目标节点号来分别，-1表示群发消息，否则为具体目标的非群发消息。

由于有非群发消息，所以为每个目标节点单独维护消息序号，用发送序号队列表示（类似于接收窗口）。

实际设计时，需要支持两种模式，一种仅支持群发的模式，一种支持非群发模式。

##### 消息接收流程

收到的消息，从消息中提取发送方身份NodeId，并将序号信息放入相应的接收滑窗。

(a)过期包(序号超出队列范畴)，则丢弃。

(b)重包(序号已在队列中，且有非空标记)，则丢弃。

(c)丢包(序号已在队列中，且有空标记)，则根据位置信息，直接把消息数据存入序列化队列。

(d)正常包(序号不在队列中，且未超出队列范畴)：

(i)该序号与节点队列的最后一个消息的序号有空隙，表示有丢包，则插入空消息到序列化队列(返回位置信息)，并将位置信息存入节点队列，做空标记。

(ii)若无丢包，将消息存入序列化队列(返回位置信息)，并将位置信息存入节点队列，做非空标记。

##### 消息重传流程

定时扫描接收端滑窗，提取空消息标记，用单播模式向该节点发送端发送消息重传请求。发送端收到消息重传请求时，从发送队列提取相应消息，并用单播模式重发该消息。

##### 异常流程

异常发生时，序列化队列将从头到尾的处理消息【忽略空消息】。有以下几种异常情况：

(A)，接收端滑窗或序列化队列满。

(B)，发现某个节点重新注册，而该节点的接收滑窗还有消息。

(C)，某节点长时间断连。

再好的序列化方案都只能保证一定时间网络断链情况，网络恢复正常后，系统的数据能一致起来。

长时间断链情况，可作为特殊情况对待，演进版本可考虑一致性检查和恢复机制。为预留这个机制，我们将在数据模型的设计上增加时间信息，一致性恢复时将以最新时间为依据。一致性检查出差异报告，可支持自动恢复和人工恢复两种模式，自动恢复的原则是：即便数据错误，也要让整个系统的数据错成一样的。

##### 定时流程

有三个定时流程。

(A)定时注册，建议1秒钟一次注册。

(B)定时检查丢包情况，发起消息重传流程。

(C)定时断线检测，发起异常处理流程。

##### 序列化能力

支持单播序列化和多播序列化两种能力，这两种能力不能在一个服务对象中同时支持。

在系统中要同时支持两种能力，就需要创建两个序列化对象。

###### 多播序列化能力

多播序列化能力分为：序列化接收和发送两种能力，这两种能力可以同时并存。

如果支持序列化接收，如果链路的对端有发送能力，那么就含有接收滑窗。如果不需要支持序列化接收能力，那么收到序列化数据消息时，直接丢弃。

如果支持序列化发送，就含有发送队列。

如果两个能力都不提供，是不能建立多播序列化服务的

###### 单播序列化能力

单播序列化能力：

序列化接收能力

序列化发送能力：

序列化可靠发送能力

序列化非可靠发送能力

序列化接收和发送两种能力是可以同时并存的，但‘序列化可靠发送能力’和‘序列化非可靠发送能力’是二选一的。

如果支持序列化接收，如果链路的对端有发送能力，那么就含有接收滑窗。如果不需要支持序列化接收能力，那么收到序列化数据消息时，直接丢弃。

如果支持序列化发送，就含有发送队列。如果支持序列化可靠发送能力，会为每个具备接收能力的链路创建发送窗口。否则仅创建一个统一的窗口。

###### 链路状态

|  |  |
| --- | --- |
| 状态 | 状态描述 |
| OnLine | 在线状态 |
| Break | 断线状态 |
| OffLine | 离线状态，离线状态下，我们会将链路删除。 |

#### 令牌模型设计

通信模块子网中有时为了某种需要，需要独占时间，进行某种事务的处理。本令牌设计模型实现1+m的能力，为了支持n+m的能力，我们设计多令牌支持方案，即n个令牌就实现了n+m的能力，这里n+m总数就是子网节点数。

本节所描述的设计方案依据RFC5059标准。

##### 令牌结构设计

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| TokenId | Uint32 | 令牌标识。 |
| DefaultNode | Uint32 | 缺省由那个节点占用。 |
| Nodes | strings | 用逗号分隔的参与竞选的节点列表。 |
| SDN | uint32 | 令牌竞选的同步范围。 |

令牌是一种资源，就像网络端口号一样，其编号必须统一定义和管理，以防不同模块之间的冲突。

##### 令牌对应用的接口。

令牌是一个自动化服务，根据配置自动启动。

采用回调接口告知应用某个令牌被抢占或放弃。

##### 令牌竞选流程



其中四个节点CPEA表示四个竞选状态（P为初始状态），有向连线表示状态迁移的驱动消息。

###### 竞选状态

|  |  |
| --- | --- |
| 状态 | 状态描述 |
| C | Candidate 候选状态。本节点是候选者。此时另外一个节点是占有令牌。 |
| P | Pending 等待状态。等待选举结果，此时本节点认为无其他节点占有令牌，但本机还未当选。 这个临时状态是为了防止竞选时出现振荡。 |
| E | Elected挂职状态。节点已当选，但未正式就职的准备状态，这个状态是为了原令牌节点做清理。 |
| A | Acknowledged就职状态。表示本节点正式占有令牌。 |

###### 驱动消息

p=Preferred BSM; 更优BSM消息

n = Non-Preferred BSM非最优BSM消息

n\*= Non-preferred BSM from Elected Owner当选Owner的非最优BSM消息

T= Bootstrap Timer timeout超时消息，不同状态下timer值可不同。

BSM(Bootstrap Message)消息定义：其实只有1条消息，但是根据消息发送者的weight和接收者的状态、接收者自己的weight、以及当前Owner的weight的对比关系，可以分为（每个节点都要记录当前Owner的地址信息和weight）：

P状态： 若‘发送者权重’> ‘自身权重’，则‘P消息’，否则‘N消息’

A、E状态：若‘发送者权重’> =‘Owner权重’，则‘P消息’，否则‘N消息’

C状态：

（1）若‘发送者权重’> =‘Owner权重’，则‘P消息’，否则：

（2）若‘发送者权重’> =‘自身权重’，则‘P消息’，否则：

（3）若‘发送者 == Owner’，则‘N\*消息’，否则‘N消息’

###### 状态签转

C候选状态 (Candidate)

1. 收到Preferred BSM 消息：

* 状态不变。
* 存储Owner信息；
* 设置BST定时器为 BS\_timeout

1. 收到BST超时消息：

* 转P状态，老Owner没动静了，我要参选！
* 设置BST定时器值为 BS\_rand\_override。

1. 收到n\*消息

* 转P状态 ，老Owner体力不支了（weight变小了，这种情况下其他成员也都知道了），或者本机自身的weight提高了（这种情况只有本机自己知道），我要参选！
* 设置BST定时器值为 （BS\_timeout + BS\_rand\_override）。

1. 收到n：（RFC中没有定义处理行为）

* 状态不变

P等选状态（Pending）

1. 收到Preferred BSM 消息：

* 转C状态
* 存储Owner信息；
* 设置BST定时器为 BS\_timeout

1. 收到BST超时消息（BS\_timeout）：

* 转E状态，当选，但不正式‘就职’
* 发送 BSM消息。
* 设置BST定时器值为 BS\_prepare。

1. 收到Non-preferred BSM：（RFC中没有定义处理行为）

* 状态不变。

E挂职状态 (Elected)

1. 收到Preferred BSM 消息：

* 转C状态 – 比自己更优的组长来了，立刻让位…
* 存储Owner信息；
* 设置BST定时器为 BS\_timeout

1. 收到BST超时消息：

* 转 A状态，Acknowledged，正式就职
* 发送 BSM消息
* 设置BST定时器值为 BS\_period。
* 将自身信息保存为Owner信息，并对应用产生竞选成功的回调。

1. 收到Non-preferred BSM：

* 状态不变
* 发送 BSM消息，有挑战者立刻镇压
* 原BST定时器不重置，继续计时

A就职状态 (Acknowledged)

1. 收到Preferred BSM 消息：

* 转C状态 – 比自己更优的组长来了，立刻让位…
* 存储Owner信息；
* 设置BST定时器为 BS\_timeout
* 对应用产生放弃令牌的回调。

1. 收到BST超时消息：

* 状态不变
* 发送BSM消息--该提醒一下各位了，我是组长
* 设置BST定时器值为 BS\_period

1. 收到Non-preferred BSM：

* 状态不变
* 发送BSM消息--不知轻重的小卒来挑战，立刻镇压

###### 竞选权重

每个节点对某个令牌设计的权重计算如下：

1. 基础权重b-weight：当前节点是该令牌的缺省节点，那么b-weight=192，否则b-weight=64
2. 参考权重r-weight：用当前节点的节点号作为参考权重。
3. 节点权重n-weight(uint64) =( ((uint64)b-weight)<<32) | r-weight。

###### 竞选定时

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 说明 |
| T0 | bs\_refer | 参考时长（以秒为单位） |
| T1 | bs\_period | 当选的Owner周期性宣告。1.5\*T0 >= T1 >= T0 |
| T2 | bs\_timeout | 成员探测Owner不可达时长。建议T2=2\*T1+ T0 = 4\*T0 |
| T3 | bs\_prepare | 挂职等待时长，该时间用于原Owner做清理工作。建议为1.5\*T0 |
| T4 | bs\_rand\_override | 从C到P后设置的定时时长。weightDelay+ addrDelay+ T0。  weightDelay = (MaxWeight – b\_weight) 0.5\*T0/10  addrDelay = myIPAddress的第三字节 / 128\*T0/10 |

换句话说竞选定时仅需要一个参考时长的配置，该时长在测试阶段可作为临时配置，系统稳定后，根据测试情况固定一个值即可。

###### 默认调整

当网管发起命令进行默认抢占令牌的节点标识变化时，

1. 当自己之前是默认的，本调整为非默认的，需要更新。
2. 当自己之前不是默认的，调整为默认的，也需要更新。
3. 否则不做任何更新。

当自己的默认角色发生改变时，需要重新计算权重，和对该令牌的T4定时时长。

##### 异常处理

当网络出现异常时，可能会引发多个子网的出现，从而导致一个令牌同时被多个节点占用。这个问题并不会引起太大的影响，只要当网络恢复正常时，会有一个收敛时间（>=T2），该收敛时间一过，是可以重新竞选出新占用者的。

#### 虚拟IP设计

这里的虚拟IP，实际上也是真实的IP地址。基于令牌技术，获得令牌的节点可以绑定相应IP地址，失去令牌的节点需要解除相应IP地址的绑定。

4.1.2章节中所涉及到的VIP，可以采用该方案进行。

特别需要提示的是G-VIP，并不是所有系统管理服务器都能与客户端连接，前面讲到我们没有划分单独的子群，我们可以在令牌的参与竞选者列表中做约束，把能与客户端连接的系统管理服务器放置到G-VIP对应的令牌竞选者列表中即可。

#### 内部信令定义

信令采用分层设计方式，这是为了：

1. 做成一个较为通用的信令模块，也便于以后的扩展。
2. 屏蔽应用不关心的内部信令。
3. 对命令字进行分段，那么各个段的命令字互不冲突。

目前分三层：

|---顶层，定义基本的UDP信令。

|---序列化信令---实现可靠传输

|---大包信令。

|---基本文件传输信令。

|---其它上层扩展的信令。

|---令牌信令（不能可靠传输，所以不能和‘序列化信令’混为一谈，故需要分层）。

##### 最大消息长度

由于采用UDP协议，需要定义最大消息长度，默认为4096(MAX\_UDP)。

##### 基本信令定义【12字节】

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| NodeNo | Uint32 | 发送节点编号： |
| PidNo | Uint32 | 发送进程编号，为了踢出重复消息。 |
| Module | uint16 | 信令服务模块号。目前预留定义【可扩展】：  0=序列化服务模块号。  1=令牌服务模块号。 |
| Cmd | uint16 | 描述消息命令字。由服务模块自行定义。 |
| Body | uint8[] | 由于是UDP消息，不做消息体长度参数。 |

PidNo的作用：

1. 收到节点号重复的消息，有两种情况：
   1. 由于组播回环【特定情况下许可】，收到自己发出的消息，是合理的，直接丢弃即可。
   2. 收到其它其它错误配置节点发出的消息，是不合理的，需要分下述两种情况分别处理：
      1. 如果系统在正常运行中，收到后来者的不合理消息，直接丢弃即可【需要保护当前正常运行的系统】
      2. 如果系统在启动过程中，收到已有节点的不合理消息，应该认为当前节点配置不正确，不允许系统启动。
2. 那如何判断是不合理消息呢？【节点号相同情况下】
   1. 如果是单播模型，由于没有回环概念，可直接认为消息不合理。
   2. 如果是组播模型，就有回环的情况，可按下述两种情况依次判断：
      1. 如果发送方的IP地址和当前节点的IP地址不同，可判断消息不合理。
      2. 如果发送方的进程号和当前节点的IP进程号不同，可判断消息不合理。

**注意本协议中使用的string类型的字段，若无特殊说明，均以0为结尾进行传输。**

##### 序列化信令定义

在基本信令定义的基础上定义四个序列化信令命令字的定义【序列化服务模块】：

|  |  |  |
| --- | --- | --- |
| 命令字 | 命令号 | 命令说明 |
| SeqRegister | 0 | 序列化模块注册信令 |
| SeqDeRegister | 1 | 序列化模块注销信令 |
| SeqData | 2 | 序列化模块数据信令 |
| SeqResendReq | 3 | 序列化模块重传信令 |
| SeqUcAck | 4 | 序列化模块单播确认信令 |
| SeqResendData | 5 | 序列化模块重传数据信令，结构与SeqData一致，当数据段为空时表示空消息，即指示SeqResendReq的失败响应 |

(1) SeqRegister消息体定义(定时发起)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| nTime | uint32 | 程序启动时间，以秒为单位，以此判断是首次注册还是后续握手。 |
| nHost | uint8[4] | 描述消息发送方主机地址，重传单播地址。 |
| nUniCast | Uint8 | 0=多播；1=单播；  注意单播节点不能向多播节点注册，反之亦然。 |
| nStatus | Uint8 | 描述当前的服务状态。0=初始状态；1=服务状态； |

(2) SeqDeRegister消息体定义 (退出发起)

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |

注销时(如系统退出)，发送该消息。如果是单播模型，需要扫描接收端节点列表，逐一发送该消息。

(3) SeqData消息体定义【12+4字节】

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Sequence | uint32 | 消息序号，发送方维护该序号的唯一性，循环递增即可。 |
| Data | uint8[] | 序列化消息包数据。Data前四个字节为序列化操作SeqOp，其它数据为序列化数据。  **模块提供序列化消息处理函数的注册，同时预留1000以内的命令字给系统自身使用。** |

序列化数据SeqOp结构定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| PlugIn | uint16 | 信令服务插件号。目前预留定义【可扩展】：  0=组包插件  1=文件传输服务插件 |
| Cmd | uint16 | 描述消息命令字。由服务模块自行定义。 |

超长包的拆包和组包机制：

1. 分包Data结构【12+4+6】

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SeqOp | Uint32 | 0，分包SeqOp |
| PackNo | uint16 | 分包的包号。 |
| Data | uint8[] | 原包数据，PackNo为0的头4字节是SeqOp |

1. 拆包原理：
   1. 如果原包数据长度大于MAX\_UDP -16字节，需要拆包。
   2. 每个分包的数据最长MAX\_UDP -22字节。
2. 自动组包原理：
   1. 设置一个组包是否失败的标记，初始为失败。
   2. 收到包号为0，如果之前的组包没有失败，则处理整个消息。清除组包错误标志，并重新开始组包。
   3. 如果出现包号不连续，标记组包失败，直到包号为0。
3. 手工组包原理【模仿流的概念】：
   1. 设置一个组包是否失败的标记，初始为失败。
   2. 收到包号为0时，回调应用读包函数，应用自行收包。
   3. 当包断序时，会返回一个接收结束的状态，直到包号为0。

(4) SeqResendReq消息体定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Package | uint32 | 需要重传的包数量，一次性可以发起多个丢包重传请求。 |
| Sequence | uint32[] | 重传包的消息序号。 |

##### 令牌信令定义【12+8字节】

在基本信令定义的基础上定义一个令牌信令命令字的定义【令牌服务模块】：

|  |  |  |
| --- | --- | --- |
| 命令字 | 命令号 | 命令说明 |
| TokenBootStrap | 0 | 令牌宣告信令 |

TokenBootStrap消息体定义：

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| TokenId | uint32 | 竞选令牌标识。 |
| Weight | uint32 | 发送者权重。 |

##### 文件传输信令

本文件传输信令实际并不遵守Ftp标准，而是自定义的私有协议。

基于序列化消息扩展文件传输信令，仅用于单播模式【文件传输服务插件】：

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作码 | 命令说明 |
| FtpOpenReq | 0 | 打开远端文件的请求 |
| FtpOpenResp | 1 | 打开远端文件的响应 |
| FtpCloseReq | 2 | 关闭远端文件的请求 |
| FtpCloseResp | 3 | 关闭远端文件的响应 |
| FtpSeekReq | 4 | 寻址远端文件的请求 |
| FtpSeekResp | 5 | 寻址远端文件的响应 |
| FtpReadReq | 6 | 读写远端文件的请求 |
| FtpReadResp | 7 | 读写远端文件的响应 |
| FtpWriteReq | 8 | 读写远端文件的请求 |
| FtpWriteResp | 9 | 读写远端文件的响应 |
| FtpTruncateReq | 10 | 截断远端文件的请求 |
| FtpTruncateResp | 11 | 截断远端文件的响应 |
| FtpMkdirReq | 12 | 创建远端目录的请求 |
| FtpMkdirResp | 13 | 创建远端目录的响应 |
| FtpRemoveReq | 14 | 删除远端路径的请求 |
| FtpRemoveResp | 15 | 删除远端路径的响应 |
| FtpListReq | 16 | 远端目录列表请求。 |
| FtpListResp | 17 | 远端目录列表响应。 |

1. **各个响应返回的响应码，在实现时，可再做调整。**
2. **如果仅考虑文件传输，上述灰色的信令都不需要实现。**

###### FtpOpen

FtpOpenReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| RemoteFile | string | 远端文件名。 |
| Mode | uint32 | （1）0x01可写，0x02尾部读写，0x04清除[只有写的情况下清除选项才有效]。以上选项可以位或运算。  （2）如果是写文件，需要自动创建文件所在目录。  （3）能写就能读。 |
| ClientSession | uint32 | 本地会话标识，支持并发操作多个文件。 |
| Transfer | uint32 | （1）0=非传输；1=文件传输；为1时，Mode仅仅为0x00下载或0x01上传。  （2）使用无需响应的FtpWrite进行传输数据，传完之后发起FtpClose并进行校验(下载时，远端发起FtpClose；上传时本端发起FtpClose)。 |

FtpOpenResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Position | int32 | <0失败，>=0表示当前位置【即大小】。 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |

注意，文件操作是基于会话管理的，当节点断开时，Ftp服务需要对相应文件会话进行处理，让上层应用感知文件操作的失败及原因。

仅支持32位的文件操作，不支持大文件模式。

###### FtpClose

FtpCloseReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| MD5 | uint8[16] | 可选参数，仅用于自动文件传输。 |

FtpCloseResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Status | uint32 | 0=成功；1=无效会话；2=MD5校验失败。 |

###### FtpSeek

FtpSeekReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Mode | int32 | -1=文件头，0=当前位置；1文件尾 |
| Offset | int32 | 偏移量 |

FtpSeekResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Status | uint32 | 0=成功；1=无效会话；2=文件损坏。 |
| Position | int32 | 当前位置。 |

同时进行读写操作，及移位操作会影响MD5的计算。当然MD5计算一般仅用于文件传输。

###### FtpRead

FtpReadReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Size | int32 | 读取大小，最大不超过MAX\_UDP-32字节=4064字节。为的是不进行拆包分包。 |

FtpReadResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Status | uint32 | 0=成功；1=无效会话；2=文件损坏。 |
| Data | uint8[Size] | MsgSize-32字节，最大4064字节。 |

###### FtpWrite

FtpWriteReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Data | uint8[Size] | MsgSize-28字节，最大4064字节。 |

FtpWriteResp操作参数定义（当做文件传输时，没有该消息）

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Status | uint32 | 0=成功；1=无效会话；2=文件损坏。 |
| Size | Int32 | 实际写入大小。 |

###### FtpTruncate

FtpTruncateReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |

FtpTruncateResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| ClientSession | uint32 | 客户端会话，即FtpOpenReq. ClientSession |
| ServerSession | uint32 | 服务端会话，打开文件成功，该字段才有意义。 |
| Status | uint32 | 0=成功；1=无效会话；2=文件损坏。 |

###### FtpMkdir

FtpMkdirReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SessionId | Uint32 | 请求会话标识 |
| Directory | string | 远端目录名 |

FtpMkdirResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SessionId | Uint32 | FtpMkdirReq. SessionId |
| Status | Uint32 | 0=成功；1=其它失败。 |

###### FtpRemove

FtpRemoveReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SessionId | Uint32 | 请求会话标识 |
| Path | string | 远端路径名，可以是文件或目录。 |

FtpRemoveResp操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| SessionId | Uint32 | FtpRemoveReq. SessionId |
| Status | Uint32 | 0=成功；2=不存在文件；3=其它失败。  对于目录，将采用rm –r模式删除。 |

###### FtpList

FtpListReq操作参数定义

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 参数说明 |
| Directory | string | 远端目录名 |
| SessionId | Uint32 | 请求会话标识 |

FtpListResp操作参数定义

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | | 数据类型 | 参数说明 |
| SessionId | | Uint32 | FtpListReq. SessionId |
| Status | | Uint32 | 0=成功；1=目录不存在；3=其它失败 |
| Count | | Uint32 | 总文件数【包括子目录数，不做递归】，可分多个包返回。 |
|  | Attr | uint8 | 0=文件，1=目录 |
|  | Name | string | 文件名 |

## 架构建议

### 系统管理服务器架构设计建议



单播序列化模块同时管理客户端和单板系统，数据传输和前转信令都走该通道。S-I-VIP和S-E-VIP两个虚拟IP对应的令牌【只有一个令牌】在内子网多播上竞选。G-VIP对应的令牌在外子网多播上竞选。系统同时仅支持一条数据传输通道。

多播序列化模块分管理范围内的【内子网】和管理范围间【外子网】。

数据管理部件，除了要考虑本地内存、文件读写外，还要考虑数据库的存储。以及进行分布式的数据管理。如果配置部分第一阶段想做的很简单，可以仅考虑故障事件的入库即通知处理。

大体启动流程：

(1)判断数据库系统的表格是否建立好，没有需要建立，SQL脚本需要考虑安全性。当没有表格时创建表格，当没有索引时建立索引等，还要考虑系统升级的字段扩展。

(2)加载本地Xml数据

先加载Xml文件，再加载操作日志。

(3)从网络子网中选择一个节点进行加载数据。【参考4.3.2.2】

(4)刷新Xml文件。

(5)设置处于服务状态。

启动一个独立的任务，处理XmlDbSync队列中的消息，当该队列为空时，转入XmlDbSync的即时处理。XmlDbSync需要做操作日志。

同时启动令牌竞选流程。

### 单板系统管理模块架构设计建议



单板系统要负责对业务系统的管理，由业务系统连接到单板系统。业务系统需要自行判断是否处于僵死状态，如果处于僵死状态，可以通过单播序列化模块主动发起注销消息。以便让单板系统管理模块感知业务系统的运行状态。

数据管理模块需要支持本地存储的数据组织[配置数据]。

大体启动流程：

1. 连接系统管理服务器(S-I-VIP)
2. 文件下载流程
3. 启动各应用[整个系统可能需要重启]

### 业务系统数据管理部件架构设计建议



数据管理模块需要考虑本地数据存储的解析，还需要考虑单板系统管理模块对业务系统的动态修改通知以及业务系统对数据的动态上报封装。

业务系统需要自行判断是否处于僵死状态，如果处于僵死状态，可以通过单播序列化模块主动发起注销消息。以便让单板系统管理模块感知业务系统的运行状态。

### 客户端数据管理部件架构设计建议



数据管理模块并不实现本地存储，仅需要实现消息及XML数据的解析和组包即可，通过单播序列化模块和系统管理服务器通信，完成各种交互(参考第3章相关章节)。

客户端向G-VIP发出GetServersRequest后，即进行串行等待。

客户端基于时间随机选择一个S-E-VIP，发送ClientLoginRequest给相应的活动系统管理服务器，即进行串行等待。

登陆成功后，启动可靠的序列化服务。

# 需求的可追踪性

## 部署需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-ARRAY-1 | 支持单域单网元部署 | 4.2.2.1 |
| REQ-ARRAY-2 | 支持单域多网元部署 | 4.2.2.1 |
| REQ-ARRAY-3 | 系统管理信令的路由 | 4.1.5，4.1.6，3.3 |
| REQ-ARRAY-4 | 系统管理平台保证系统管理信息在平台内传输的可靠性 | 4.2.2 |
| REQ-ARRAY-5 | 系统管理平台内的系统信息同步 | 3.3 |
| REQ-ARRAY-6 | 系统管理平台支持多客户端登录 | 3.6 |
| REQ-ARRAY-7 | 系统管理平台支持客户端同时多点连接 | 3.6 |
| REQ-ARRAY-8 | 系统管理平台内安全信息同步 | 3.3 |

## 数据配置功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-CONFIG-1 | 系统管理平台对外提供统一的配置数据访问与操作功能 | 4.1.6  3.6 |
| REQ-CONFIG-2 | 统一的系统管理配置处理逻辑，不受新增配置业务影响 | 4.1 |
| REQ-CONFIG-3 | 系统管理平台支持不同数据格式配置需求 | 4.1 |
| REQ-CONFIG-4 | 系统管理平台与客户端之间配置数据交互 | 4.1.6  3.6 |
| REQ-CONFIG-5 | 系统管理平台与业务应用之间配置数据交互 | 4.1.6  3.7 |
| REQ-CONFIG-6 | 系统管理平台支持XML数据解析 | 4.1 |
| REQ-CONFIG-7 | 系统管理平台具备配置数据存储功能 | 3.5 |
| REQ-CONFIG-8 | 系统管理平台支持业务配置数据同步 | 4.2.2  3.3.1 |
| REQ-CONFIG-9 | 系统管理平台支持配置数据校验 | 4.1.3 |
| REQ-CONFIG-10 | 系统管理平台支持配置数据回滚功能 | 由上层应用支持 |
| REQ-CONFIG-11 | 配置数据备份与恢复 | 由上层应用支持 |

## 故障管理功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-ALARM-1 | 故障的上报通道 | 由上层应用进一步支持，可以定义特定的故障表格，对于该协议接口，全部统一为表格操作。 |
| REQ-ALARM-2 | 故障信息定位功能 |
| REQ-ALARM-3 | 故障信息识别功能 |
| REQ-ALARM-4 | 故障信息保存及维护 |
| REQ-ALARM-5 | 故障信息检索 |
| REQ-ALARM-6 | 故障信息备份功能 |

## 业务管理功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-SERVICE-1 | 系统管理平台提供业务管理通道 | 4.1.3.2，采用‘控制类型’字段用于实现业务控制功能。 |

## 系统监控功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-MONITOR-1 | 系统管理平台支持对单板与进程进行监控 | 3.4.2定义“16=监控”，实现自动监控功能。 |
| REQ-MONITOR-2 | 系统管理平台监控功能上报故障信息 |
| REQ-MONITOR-3 | 系统管理平台支持对监控门限值进行配置 |
| REQ-MONITOR-4 | 系统管理平台支持对监控结果进行查询 |

## 系统信息上报功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-EVENT-1 | 事件信息上报 | 由上层应用进一步支持，可以定义特定的故障表格，对于该协议接口，全部统一为表格操作。 |
| REQ-EVENT-2 | 事件信息的保存与维护 |
| REQ-EVENT-3 | 事件信息检索 |
| REQ-EVENT-4 | 事件信息备份功能 |

## 软件管理功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-SOFT-1 | 系统管理平台软件管理功能 | 4.2.2.5.5提供FTP信令支持，  3.2.1提供应用升级、启动的基本信令支持。 |
| REQ-SOFT-2 | 系统管理平台自身软件故障处理 |
| REQ-SOFT-3 | 系统管理平台业务软件的启动管理 |
| REQ-SOFT-4 | 系统管理平台业务软件启动过程管理 |
| REQ-SOFT-5 | 系统管理平台监控业务软件运行 |
| REQ-SOFT-6 | 系统管理平台处理业务软件运行异常 |
| REQ-SOFT-7 | 系统管理平台支持软件管理策略配置 |
| REQ-SOFT-8 | 系统管理平台支持软件结束通知 |
| REQ-SOFT-9 | 系统管理平台支持文件下载机制 |
| REQ-SOFT-10 | 系统管理平台支持业务软件的更新 |
| REQ-SOFT-11 | 系统管理平台支持客户端对业务软件运行控制功能 |

## 系统安全功能需求

| **需求 ID** | **需求标题** | **对应章节** |
| --- | --- | --- |
| REQ-SAFE-1 | 系统管理平台提供对用户权鉴机制 | 3.4.1 |
| REQ-SAFE-2 | 系统管理平台支持灵活的用户权鉴方式 | 3.4.2 |
| REQ-SAFE-3 | 系统管理平台灵活的用户权限设定 | 3.4.2 |
| REQ-SAFE-4 | 系统管理平台的用户权限告知 | 3.4.1 |
| REQ-SAFE-5 | 客户端登录后的信令安全保证 | 3.4.3 |
| REQ-SAFE-6 | 用户操作日志自动记录 | 由上层应用支持 |
| REQ-SAFE-7 | 系统管理平台支持安全配置 | 由上层应用支持 |

# 注释

|  |  |
| --- | --- |
| 缩略语 | 原语及说明 |
| NMS | Network Manage System，网络管理系统 |
| SNMP | Simple Network Manage Protocol，简单网络管理协议 |
| TR069 | 基于SOAP的远端设备管理协议，不仅仅是通信协议，同时也定义了管理模型。 |
| SMP | System Manage Platform，系统管理平台 |
| SMS | System Manage Server，系统管理服务器 |
| SMC | System Manage Client |
| SMU | uni-board System Manage Unit，单板系统管理模块 |
| SMD | System Manage Domain，系统管理范围 |
| MDN | Manage Domain Number，管理范围编号 |
| MUN | Manage Unit Number，管理模块编号 |
| SDN | Synchronization Domain Number，同步范围编号 |
| SSN | Sub System Number，子系统编号 |
| OMI | Object Managed Identity管理对象标识 |
| MMS | MDN-MUN-SSN，管理对象位置标识。 |

阴影部分是标准命名，其它是本协议接口定义的私有命名。