

|  |  |  |  |
| --- | --- | --- | --- |
| 活动编号（ID）： | | | |
| 项目阶段 | ■计划阶段 | □需求阶段 | □概设阶段 |
| □详计阶段 | □编码阶段 | □单元测试阶段 |
| □集成测试阶段 | □系统测试阶段 | □关闭阶段 |
| 产品名称 | | 产品型号/版本 | 总页数 |
| 平台2.0 | | V2R1 | 共 ×× 页 |

CUI概要设计

（仅供内部使用）

|  |  |
| --- | --- |
| 文 件 编 号： | KIPD-MB-0001 |
| 版 本 号： | V2R1 |
| 实 施 日 期： | 2011-11-1 |
| B M 等 级： | ■M\*M □JI\*M □JUE\*M |
|  | |
| 编 制： |  |
| 审 核： |  |
| 会 签： |  |
|  |  |
|  |  |
| 批 准： |  |

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| 2011-9-13 | 0.1 | 初稿完成 | 梁黎 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 前言 5](#_Toc333393910)

[2 范围 5](#_Toc333393911)

[2.1 项目名称 5](#_Toc333393912)

[2.2 项目目标 5](#_Toc333393913)

[2.3 项目时间 5](#_Toc333393914)

[3 定义 5](#_Toc333393915)

[4 运行环境 5](#_Toc333393916)

[5 主要功能 6](#_Toc333393917)

[5.1 业务层次图 6](#_Toc333393918)

[5.2 业务功能分析 6](#_Toc333393919)

[5.2.1 CUI在2.0版本中的新特性 6](#_Toc333393920)

[5.2.2 WebService客户端接入 6](#_Toc333393921)

[5.2.3 CU的会话维护 7](#_Toc333393922)

[5.2.4 CU的事务管理 7](#_Toc333393923)

[5.2.5 CU的保活机制 7](#_Toc333393924)

[5.2.6 CU的消息推送机制 8](#_Toc333393925)

[5.2.7 CUI与CMU的消息交互 8](#_Toc333393926)

[5.2.8 CU的负载均衡 8](#_Toc333393927)

[5.2.9 统计CU在线数 8](#_Toc333393928)

[5.2.10 接入网管系统 8](#_Toc333393929)

[6 架构设计 9](#_Toc333393930)

[6.1 CUI框架设计 9](#_Toc333393931)

[6.2 RPC模型 9](#_Toc333393932)

[6.3 CUI架构描述 10](#_Toc333393933)

[7 CUI流程描述 10](#_Toc333393934)

[7.1 CU登录流程 10](#_Toc333393935)

[7.2 CU保活流程和消息推送流程 12](#_Toc333393936)

[7.3 CU请求码流 13](#_Toc333393937)

[7.4 CU操作外设流程 15](#_Toc333393938)

[8 云系统与CU漫游登录 15](#_Toc333393939)

[8.1 CU的会话管理与消息路由 16](#_Toc333393940)

[9 模块框图 16](#_Toc333393941)

[10 断链处理 17](#_Toc333393942)

[10.1 CUI与Proxy断链 17](#_Toc333393943)

[10.2 CUI与CMU断链 17](#_Toc333393944)

[10.3 CUI与CU断链 17](#_Toc333393945)

[10.4 CUI与3AS断链 17](#_Toc333393946)

[10.5 CUI与外设模块断链 18](#_Toc333393947)

# 前言

平台2.0中，CU采用WebService接口与平台交互，平台内部模块都采用SIP协议相互通信。CUI起到两种协议相互转换的作用。WebService的RPC调用是阻塞式的，而SIP协议是异步消息，CUI的设计模型必须满足这种应用场景。

# 范围

## 项目名称

科达视频监控平台ViewShot V2R1

## 项目目标

满足监控平台2.0的需要。

## 项目时间

随平台2.0一起发布。

# 定义

CUI：平台2.0中的负责接入CU的协议转换模块

CU：用户客户端。

# 运行环境

CUI：Linux

# 主要功能

## 业务层次图



图 5-1 CUI业务层次图

## 业务功能分析

### CUI在2.0版本中的新特性

平台2.0中，CU采用WebService和CUI交互，WebService有三大特点：1，WebService本身是无状态和无连接的协议。2，只有简单的请求应答消息模型。3，RPC阻塞调用。

基于这些新特性，在平台2.0中CUI不再是简单的透传CU消息，而是作为CuAgent代替CU处理一些平台消息，或者代替CMU处理一些CU消息。所以CUI在某种程度上就是CuAgent，实现时要站在CU的位置上思考。

CUI模块可以简单划分为前端接入和后端处理两部分，前端接入模块叫做cui\_fcgi，后端处理叫做cui\_server。

### WebService客户端接入

WebService是阻塞式的RPC调用，所以CUI必须在Serve一个SOAP请求的函数内将所有请求投递到服务器端（例如CMU，3AS），并且阻塞等待后将收集的结果返回给函数的OUT参数。同时，CUI不止处理一个CU请求，而是同时处理很多CU的RPC请求，必须对每一个请求都先阻塞等待，然后CUI后端与服务器端异步通信收集结果。

Cui\_fcgi的第一要务就是建立一个阻塞调用模型。该模型允许CUI同时处理多个CU请求，对每个请求都先阻塞等待，并在收集完结果之后再解除阻塞。

### CU的会话维护

WebService是无连接协议，CU和CUI之间没有对应的OspNode，CUI也无法从请求中判断是来自哪个CU的，因此要有CU会话状态管理。

在CU登录请求时，为CU分配一个SessionID，并唯一对应一个事务对象。该事务对象的生命周期与CU的登录周期一样长，在CU登录时创建，登出时销毁。这个事务对象就叫做CuAgent，它就是CU在CUI内的代理，所有的CU请求都由代理发送出来。

CuAgent的TaskID与CU的登录SessionID唯一绑定，同时负责CU的事务管理与保活。

### CU的事务管理

当收到CU的登录请求之后，CUI就创建了CuAgent事务，并由CuAgent向服务器端发送登录请求，CMU会在登录应答中附带SessionID。CUI将CuAgent的TaskID与SessionID绑定，并将SessionID回复给CU保存。

后续CU再发起新的请求时必须附带SessionID，CUI通过SessionID找到CuAgent事务，这些请求由CuAgent处理并创建新的事务。这些事务不仅由CUI管理，同时也在CuAgent的任务集合中。CMU回复消息时，则直接通过TaskID找到新创建事务，交由新创建的事务处理。

在CUI中，所有的资源都是由事务Task掌控的，只要删除CuAgent并连带删除其任务集合中的所有Task事务，CU的所有分配资源都会回收，不存在资源泄漏问题。

CUI收到CMU的主动通知的消息时，首先通过SessionID找到CuAgent，然后交由CuAgent处理该消息。由于WebService只有请求应答模式，没有逆向请求应答以及主动通知模式。所以这些主动通知消息被CuAgent缓存，等待CU轮询时再回复。

CUI与服务端之间的会话协议需要满足事务三元组<SessionID，TaskID，SeqNum>，在SIP会话中SeqNum和TransID是等价的。

### CU的保活机制

WebService是无连接的协议，而业务上CUI必须知道CU是否在线，以此决定是否继续保留为该CU分配的资源。所以CUI业务层上必须有对CU的保活机制。

CU必须定期向CUI发送保活请求，CuAgent中设置了一个超时计数，每次收到保活请求后就计数清零。一旦计数超时CUI就会认为CU断链，将会通知CMU并释放CuAgent，连带释放其任务集合。

注意：CUI对于CMU来说是透明的，或者说对于CMU来说，CUI就是CU的Agent。CUI作为SIP的UA注册到CMU上，CMU只对CUI设置心跳，而不对CU设置心跳。一旦CU保活失败，CUI会向CMU发送CU断链通知。

### CU的消息推送机制

WebService只有简单的请求应答操作，而CMU需要向CU主动发送一些通知消息，例如PuStatus通知。CUI必须具有消息推送机制。

CMU发送的主动通知消息中，根据SessionID可以找到CuAgent，CuAgent会收集需要主动通知的消息并缓存下来。CU被要求定期发送保活请求，除了刷新CuAgent中的超时计数外，CUI还会告诉CU当前哪些请求通知可用。这些请求通知会整理成一个列表，告诉CU需要马上发送哪些WebService请求。

当CUI收到重复的CMU通知消息时，CuAgent不判断消息是否重复，所有的消息都当做string被CuAgent缓存到一个唯一的队列。当CU获取了缓存通知后，CuAgent会删除这些缓存消息。

### CUI与CMU的消息交互

Cui中的一个APP要负责与CMU通过SIP协议进行异步通信。该APP是cui\_server的核心处理线程，所有的事务都由该APP处理。每个事务负责处理异步消息。

### CU的负载均衡

在2.0系统中，CUI是通过WebService无连接协议，所以整个系统只有一个CUI后端，但是有多个fastcgi前端负责接入CU的WebService请求。

所谓的负载均衡就是apache的Web服务器自动将多个WebService请求发送到空闲的fastcgi上。

在服务器级别的负载均衡。CU登录后获取CUI列表，当某个CUI的CU在线数达到最大值时，就返回一个错误码。CU就尝试列表中下一个CUI登录。CU每次登录成功后就更新CUI列表并永久化到本地保存。

### 统计CU在线数

CU认证成功后，CUI会通知守卫线程，网管还有LDS，告知CU登录人数变化。

### 接入网管系统

CUI通过NmsAgent与网管系统交互。主要是向网管提供当前平台运行信息，例如CPU，内存状态，当前用户接入数等。

目前CUI暂不接入Nms，由CMU搜集CUI数据并返回给NMS。

# 架构设计

## CUI框架设计



图 6-1 CUI框架设计

## RPC模型

同步接口服务一般分为两层，1，API层；2，消息交互层。原理如下：

1. 用户调用API接口，在API层获得一把锁，然后将API函数参数转化为请求消息发送出去。然后API层加锁阻塞，等待结果回复。
2. 消息交互层异步处理所有消息，开始完整的请求应答流程，并等待结果响应。同时轮询处理超时事务。
3. 如果事务收到操作结果，则在API层的对象中填入结果，再释放锁。事务的生命周期终结。
4. 如果事务超时，同样在API层得对象中填入超时失败结果，再释放锁。事务的生命周期终结。
5. API函数被解锁继续执行。从结果对象中取出操作结果，再填入API的OUT参数。

但是在cui\_fcgi中情况略有不同。每个cui\_fgci同一时刻只服务于一个CU的WebService请求，因此没有必要单另起一个消息交互的线程，可以直接在API层中阻塞时不断轮询操作结果，如果轮询超时则认为处理失败。

## CUI架构描述

cui\_fgci和cui\_server之间的通信协议必须设计的足够简单，cui\_fgci发送一条请求消息，收到一条应答消息就能够获取所有的期望操作结果。复杂的业务处理都在cui\_server端完成。

如图6-1所示，CU的WebService请求通过Apache服务器转给一个空闲的cui\_fgci，然后由cui\_fgci将WebService方法中的函数参数转为一条SIP请求消息体，再发送给cui\_server。

在cui\_fgci发送请求之后就阻塞等待处理结果。

cui\_server收到了请求之后会通过SessionID首先找到CU对应的CuAgent，然后创建一个新的请求事务。由请求事务发送SIP消息给CMU。等到CMU应答了之后，cui\_server再将结果整理成一条应答消息发送给cui\_fgci。

cui\_server是异步处理所有SIP消息的。它以CU的身份发送所有请求。

cui\_fgci收到处理结果后就解除阻塞，并将结果填写到WebService方法的OUT参数中。

# CUI流程描述

下面我们描述几个常用的消息流程。

## CU登录流程

CU的登录分为三个步骤：1，登录认证；2，获取设备列表；3，获取其他平台数据。这里CU登录时不再刷新PU状态，改由登录成功后主动订阅PU状态。CU在整个登录过程中只身份认证一次。

图 7-1 CUI登录时序图

如图7-1所示，CU的登录步骤如下：

1. CU上输入用户名Liangli@Nanjing和密码，并选择登录的平台地址。用户如果需要漫游登录Suzhou，则输入漫游平台Suzhou的地址即可。点击登录按钮后CU调用WebService的登录方法将请求发送给Suzhou。
2. Suzhou的CUI收到WebService请求，向Nanjing的3AS发送身份身份认证请求。返回的结果包括Liangli@Nanjing的用户基本信息，是否允许多人登录，用户设备修改时间戳等。
3. CUI@Suzhou随即向CMU@Suzhou发起CU登录请求。由于CMU信任CUI，因此CMU直接创建CU事务，生成一个SessionID与该事务绑定，并返回SessionID和登录结果。  
   3AS@Nanjing不能防止Liangli@Nanjing多用户登录，所以只能由CMU@Suzhou或者CMU@Nanjing来判断是否允许多用户登录了。
4. CUI@Suzhou将SessionID和CuAgent绑定，并向CU返回登录结果，包含CMU为CU分配的唯一SessionID。此SessionID和Liangli@Nanjing用户所在域无关。后续该用户都使用SessionID@Suzhou作为唯一标识。
5. CU请求刷新设备列表。CUI@Suzhou向3AS@Nanjing发送此请求，并得到设备列表和权限信息应答。  
   CUI@Suzhou还可以缓存Liangli@Nanjing用户的设备列表和权限信息，并实时接收3AS@Nanjing的设备更新通知。一旦有新的Liangli@Nanjing登录，CUI@Suzhou可以直接返回其用户设备列表和设备权限。  
   同时CU也可以根据用户设备修改时间戳判断是否需要刷新设备列表，还是使用CU本地缓存。
6. CU请求获取用户数据。CU需要获取一些平台数据，例如“CU自定义数据”等都是存在用户所在域。CUI@Suzhou向CMU@Suzhou获取这些用户数据。同理，CU保存或者修改用户数据时，CUI@Suzhou也想CMU@Suzhou发送这些请求。任何平台模块都需要先注册再获取资源，所以不允许CUI@Suzhou直接向CMU@Nanjing发送消息，必须通过CMU@Suzhou走跨域转发。

## CU保活流程和消息推送流程

CUI需要完成心跳保活和消息推送两种功能。这两种功能都通过CU定期发送保活请求实现。



图 7-2 CU保活时序图

步骤如下：

1. CU定期发送保活请求。此时CUI只是简单的应答。（由于心跳保活请求的频率直接影响CU接收通知消息的响应时间，太短消耗平台性能，太长则响应过慢。则将保活轮询的代价放到CUI，节省WebService通信代价；CUI收到保活请求后，阻塞1秒钟后返回，如果此期间内有通知则立刻通知，否则1秒超时后返回。CU在保活请求应答后，可立刻再发起新的保活请求。CU有单独线程进行保活）
2. CUI定期进行超时计数轮询。CUI负责管理所有登录的CU事务CuAgent，并对事务CuAgent进行超时计数TimerPoll。如果事务计数超时了，则认为该事务已经无效。
3. CUI接收平台的通知消息并缓存在CuAgent中。
4. 当再有CU的保活请求到来时，CUI发现CuAgent中有缓存消息，则在应答时告诉CU有通知消息可用，并告诉CU这些通知消息的方法。
5. 当有重复的CMU通知消息时，CuAgent会覆盖老的通知消息，只保存最新的通知消息。
6. 当CU调用了通知请求方法，CuAgent会删除缓存消息。

## CU请求码流



图 7-3 CU视频浏览时序图

这一流程主要描述CUI处理一般操作的内部流程，步骤如下：

1. CU请求视频浏览。
2. Apache收到此请求后，将请求转发给空闲cui\_fcgi进程。如果没有空闲cui\_fcgi，则Apache会尝试创建空闲cui\_fcgi进程。如果进程池已经达到最大数，则直接返回系统繁忙给CU。
3. cui\_fcgi将请求转化成异步消息给cui\_server，然后加锁，等待结果返回。
4. cui\_server收到请求收，首先查找Cu对应的CuAgent。将请求交由CuAgent处理。
5. CuAgent创建一个视频浏览事务，保存cui\_fcgi的连接句柄，并向CMU发送视频浏览请求。此消息也是异步的，含带浏览事务号。
6. CMU返回视频浏览成功给cui\_server。cui\_server依据事务号找到对应的视频浏览事务。并将该应答消息丢该事务处理。
7. 视频浏览事务判断本次事务生命周期是否终结。如果是最后一条期望应答消息到来，则将操作结果写入cui\_fcgi的连接句柄中。如果应答消息超时，则直接写入失败结果。
8. cui\_fcgi从连接句柄中读取应答操作应答。并将结果填入WebService的函数OUT参数中。函数返回。
9. CU收到视频浏览成功的结果后，通知CU上层创建解码器准备接收码流，然后向WebService接入线程发送请求，通知目的准备好接收。
10. 剩余流程同上。但是需要注意的是，CU的两次WebService请求并不由同一个cui\_fcgi处理，因此cui\_server不能依据cui\_fcgi来标识CU，cui\_fcgi本身也不缓存任何状态。

## CU操作外设流程



图 7-4 CU操作外设流程

如图7-4所示，CU的操作顺序如下：

1. CU向CUI发起获取外设数据信息请求。
2. CUI向XXXAgent发起注册请求。
3. XXXAgent发起注册响应给CUI。
4. 注册成功后，CU以CUI身份，由CUI向XXXAgent发送获取外设数据请求。
5. XXXAgent向CUI发送获取外设数据响应。
6. CUI将消息转发给CU。

# 云系统与CU漫游登录

在云系统中，CU可以在任意云端登录，且享受如同本地登录的服务。但是CU的数据终究是要存储在一个地方，并且系统知道在哪里获取。理所当然，CU的数据应该存储在CU所属的域。

CU本身不需要关心这些数据存储在哪里，它只需要调用当前登录域的WebService方法，并得到应答。例如Liangli@Nanjing漫游登录到Suzhou，CU只向Suzhou的Apache发起WebService请求就能够获取到存储在Nanjing域的数据了。

根据“在获取资源前需要先注册”的原则，CUI@Suzhou不允许直接访问CMU@Nanjing，所有消息都必须通过CMU@Suzhou中转。

但是有些消息是CUI直接发送到目标服务器的，例如CUI注册到3AS@Nanjing上，直接向3AS@Nanjing发送CU登录请求。

## CU的会话管理与消息路由

在5.2.3 CU的会话维护章节我们讲到，因为WebService是无连接协议，所以需要为每一个登录的CU都分配一个SessionID并唯一绑定一个CuAgent事务，所有的WebService都包含此SessionID，CUI管理所有的CuAgent，并根据SessionID找到CuAgent并将请求交给其处理。

在云系统中，CU可以在任意一个域漫游登录，例如Liangli@Nanjing漫游登录到Suzhou域。那么Suzhou的CMU为Liangli@Nanjing分配的SessionID可能为abc123@Suzhou。abc123@Suzhou是作为业务路由而使用，而Liangli@Nanjing是作为日志记录使用。虽然OspSip库中都是根据TransID回复应答消息，但是难保出现A发送消息给B，B转发消息给C，而C直接给A回复消息的情况。不过根据SIP的精神，我们应该尽量避免这种情况发生。

# 模块框图

* Cui\_fcgi：WebService的接入端，负责将WebService接口中的消息转换为xml数据发送给cui\_server。
* Cui\_server：CUI中的业务实现端，收到cui\_fcgi请求后生成事务对象，处理请求完毕后将结果返回给cui\_fcgi。
* OSP：包括Osp和OspExt库，Osp提供Telnet调试打印功能以及对操作系统接口的封装。OspExt提供事务模型。
* Sip协议栈：包括OspSip和kdsip，kdsip由组件组提供并维护，实现SIP协议交互。OspSip对其UA功能进行封装，增加易用性。
* SOAP协议栈：实现WebService，由Apache调用。
* TCP协议栈：实现cui\_fcgi和cui\_server之间消息传递。



图 1-1 DNS View模块

# 断链处理

## CUI与Proxy断链

1. CUI在启动时会向Proxy注册，当注册成功后才会进入Server状态。
2. 当CUI与Proxy断链后，会释放所有资源，所有任务都会释放掉
3. CUI会回到idle状态并立刻尝试重新注册Proxy。

这里涉及到一个问题，SIP是无连接的，CUI和Proxy断链，但是可能瞬间就又注册成功了，而CMU向CUI发送HB请求只是超时一次，并不感知CUI断链过；又或者CUI进程挂掉后又重新注册到Proxy上，CMU同样感知不到CUI断链。针对这两种情况，建议OspSip的心跳hb\_option消息中增加UUID。

根据评审会议，UA模块不处理Proxy断链。

## CUI与CMU断链

CUI在注册到Proxy后向CMU注册，注册成功后进入服务态才会允许CU注册。

当CUI与CMU断链后，所有的CU的事务都销毁，并通知CU登出和释放所有资源，CUI重新进入IDLE状态，此状态下事务不接受任何CU请求，一律拒绝。

## CUI与CU断链

CUI对每一个CU都有一个CuAgent事务的会话管理，并对CU超时计数。一旦计数超时导致断链，CUI会释放所有CU相关的Task。

## CUI与3AS断链

当CU登录时，CUI会向3AS注册，查询用户信息缓存在本地，同时接收3AS的设备通知消息，并更新到用户信息中。当3AS断链后，CUI销毁此用户信息。

当CU再次登录时，CUI再尝试注册3AS，注册失败则拒绝CU登录。

## CUI与外设模块断链

CUI与某外设模块断链，中断与该外设模块的所有业务，并删除相应的事务以及事务管理表中的数据。当CU再次发起请求时则进行注册，如果注册不上则返回相应的错误码。