分布式系统

应用架构设计

编制： 王 科 威

审核：

版本： V1.0.0

2020年5月18日

目录

[修改历史 I](#_Toc41290392)

[一 引言 1](#_Toc41290393)

[1.1 编写目的 1](#_Toc41290394)

[1.2 规范说明 1](#_Toc41290395)

[二 定义 1](#_Toc41290396)

[2.1 协议格式 1](#_Toc41290397)

[2.2 协议定义 1](#_Toc41290398)

# 修改历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **说明** | **作者** | **审核** | **日期** |
| V1.0.0.0 | 初稿 | 王科威 |  | 2020-05-25 |

# 引言

## 编写目的

在视频数据越来越多的情况下，如何做到投入最少的IT资源、提供更高的用户数据和分析能力？分布式系统应用架构即是通过采集这些非结构化数据源，并将它们汇聚到一起存储到公有或私有云上，帮助用户思考能用数据做什么？从中获取有价值的数据及知识，并有可能在未来提供现在还无法想象的云服务。

## 规范说明

### 命名规范

|  |  |  |
| --- | --- | --- |
| 编号 | 名称 | 说明 |
| 1 | XMQ | 消息分发服务 |
| 2 | CMS | 资源管理服务 |
| 3 | WEB | WEB后台服务 |
| 4 | DLB | 视频监控设备接入负载均衡服务 |
| 5 | HKD | 海康视频监控设备接入服务 |
| 6 | DHD | 大华视频监控设备接入服务 |
| 7 | SLB | 流媒体负载均衡服务 |
| 8 | XMS | 私有流媒体分发服务 |
| 9 | RTSP | RTSP流媒体分发服务 |
| 10 | RTMP | RTMP流媒体分发服务 |
| 11 | HLS | HLS流媒体分发服务 |
| 12 | GB28181 | GB28181服务 |
| 13 | DSS | 数据同步服务 |
| 14 | AIS | AIS串口服务 |
| 15 | MCS | 母钟串口服务 |

### ID标识规范

所有ID标识使用UUID规范。

### 用户数据规范

用户数据格式按照Google protocol buffers开源项目的sytex2格式要求进行串行化处理。

# 系统架构设计

## 系统架构图



## 系统结构图说明

### 平台应用层

### 平台处理层

### 分布式服务层

#### 消息分发服务（XMQ）

消息分发服务是分布式服务层中所有服务间网络通信的桥梁。

* 该服务不解析任何用户数据，只负责在各个服务间转发消息。
* 该服务作为桥梁，同时面向两侧用户，一个是Worker用户，另一个是Client用户。

*消息分发服务在分布式服务层中是唯一的。*

#### 资源管理服务（CMS）

资源管理服务是分布式服务层中所有服务的管理工具。分布式服务层中的服务以名称作为标识向资源管理服务注册、心跳和注销，该服务自动监视各服务的在线状态，超过90s没有状态更新的服务作离线处理。资源管理服务支持服务在线状态查询功能。

*资源管理服务在分布式服务层中是唯一的。*

#### WEB后端服务（WEB）

*WEB后端服务同时具备Worker模型和Client模型二者的特性。*

#### 视频监控设备接入负载均衡服务（DLB）

该服务每间隔5s向资源管理服务查询在线的视频监控设备接入服务。在每次设备配置时选取出最合适的处理者。

*视频监控设备接入负载均衡服务在分布式服务层中是唯一的。*

#### 海康视频监控设备接入服务（HKD）

#### 大华视频监控设备接入服务（DHD）

#### 流媒体负载均衡服务（SLB）

该服务每间隔5s向资源管理服务查询在线的流媒体分发服务。在每次流配置时选取出最合适的处理者。

*视频监控设备接入负载均衡服务在分布式服务层中是唯一的。*

#### 私有流媒体分发服务（XMS）

#### RTSP流媒体分发服务（RTSP）

#### RTMP流媒体分发服务（RTMP）

#### HLS流媒体分发服务（HLS）

#### GB28181服务（GB28181）

#### 数据同步服务（DSS）

#### 串口设备接入服务（AIS & MCS）

*串口设备接入服务在分布式服务层中可以多次注册，但名称必须不同。*

* AIS
* MCS

### 分布式数据存储层

### 数据ETL层

## 分布式服务层网络架构图



## 分布式服务层网络架构说明

分布式服务层网络架构由3种不同的角色组成，分别为：Dispatcher、Worker和Client。Dispatcher使用桥接的思想在逻辑上将Worker和Client分离成面向内部服务和外部的不同服务，并采用不同的数据格式进行交互和转换。Worker之间和Worker与Client之间也可以直接使用发布订阅的方式直接进行数据交互。

### Dispatcher模型

首先，Dispatcher只负责数据的交换，就如同邮局一样，Dispatcher只按照发送ID和接收ID进行数据转发，不解析任何数据内容。

其次，从逻辑结构上来说，Dispatcher可以是点对点的互联，也可以是上下级的级联。无论是互联还是级联结构，Dispatcher模型都必须具备Worker模式的特性才能实现。

### Worker模型（面向内部）

Worker模型是分布式服务层网络架构中面向内部的服务。与Client模型不同之处在于Worker模型必须实现注册、心跳和注销等业务逻辑。

### Client模型（面向外部）

Client模型是分布式服务层网络架构中面向外部的服务。

### Pub模型

Pub模型即可以向分布式服务层网络结构的内部发送订阅数据，也可以向分布式服务层网络结构的外部发送订阅数据。

### Sub模型

## 分布式数据处理层架构图

## 分布式数据处理层架构说明

# 分布式服务层通信协议

基于系统中存在的3中不同角色，以XMQ为桥，通信协议分为Client端使用的面向外部服务协议和Worker/XMQ端使用的面向内部服务协议：

## XMQ <--> Client

### 协议格式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***CommID*** | ***Placeholder*** | ***RoleID*** | ***Via*** | ***UserData*** |

### 数据定义

* CommID：Client端通信ID标识。
* Placeholder：空。
* RoleID：Client端角色ID标识，即“Client”。
* Via：消息路径。

1. 消息路径由Worker和XMQ的服务名称组成，服务名称通过Client端向XMQ查询获得。
2. 消息路径为空表示Client端向XMQ查询有那些Worker服务在线。
3. 当消息路径由一个或多个“：”分隔时表示数据下行转发的路径，转发路径从当前XMQ的下一个XMQ节点开始。

* UserData：用户数据。

## XMQ <--> Worker

### 协议格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***CommID*** | ***Placeholder*** | ***RoleID*** | ***From*** | ***Via*** | ***Data*** |

### 数据定义

* CommID：Worker端通信ID标识。
* Placeholder：空。
* RoleID：Worker端角色ID标识，即“Worker”。
* From：数据发送端CommID标识。
* Via：消息路径。

1. 消息路径由Worker和XMQ的服务名称组成。
2. 消息路径为空表示Worker端向XMQ注册或心跳。
3. 当消息路径由一个或多个“：”分隔时表示数据下行转发的路径，转发路径从当前XMQ的下一个XMQ节点开始。

* Data：数据。

## XMQ <--> XMQ

与XMQ <--> Worker相同。

## Client <--> Worker

系统内部的部分Worker应用服务支持Client端进行数据订阅。数据格式由用户确定。订阅的地址在Client端查询Worker应用服务时获取。

# 内部通信时序图

## Client <--> XMQ

### 服务查询



Data：”Query”表示查询请求，以”;”分隔的服务名称表示查询应答。

## Client <--> XMQ <--> XMQ

### 服务查询



## Client <--> XMQ <--> Worker

### 数据转发



### 数据订阅



## Client <--> XMQ <--> XMQ <--> Worker

### 下行转发



## Worker <--> XMQ

### 服务注册/心跳



* Name：Worker服务名称。
* Data：”Register / Heartbeat”。

### 服务查询



* Name：Worker服务名称。
* Data：”Query”表示查询请求，以”;”分隔的服务名称表示查询应答。

## Worker <--> XMQ <--> Worker

### 数据转发



## Worker <--> XMQ <--> XMQ <--> Worker

### 上行推送

# 应用层通信协议格式

各类消息按照统一的协议格式进行封装和解析，协议格式定义如下：

syntax = "proto2";

import "Alarm.proto";

import "Algorithm.proto";

import "Component.proto";

import "Crew.proto";

import "Device.proto";

import "Status.proto";

import "User.proto";

package msg;

message MSG

{

enum Type

{

MSG\_NONE = 0;

ALARM = 1;

ALGORITHM = 2;

COMPONENT = 3;

CREW = 4;

DEVICE = 5;

STATUS = 6;

User = 7;

}

required Type type = 1 [default = MSG\_NONE];

required int32 sequence = 2;

required int32 timestamp = 3;

optional AlarmMsg alarmmsg = 4;

optional AlgorithmRequest algorithmrequest = 5;

optional AlgorithmResponse algorithmresponse = 6;

optional ComponentRequest componentrequest = 7;

optional ComponentResponse componentresponse = 8;

optional CrewRequest crewrequest = 9;

optional CrewResponse crewresponse = 10;

optional DeviceRequest devicerequest = 11;

optional DeviceResponse deviceresponse = 12;

optional StatusRequest statusrequest = 13;

optional StatusResponse statusresponse = 14;

optional UserRequest userrequest = 15;

optional UserResponse userresponse = 16;

}

# 应用层通信协议定义

## 组件类

#### 协议定义

syntax = "proto2";

package msg;

message ComponentRequest

{

enum Type

{

XMQ = 1;

WEB = 2;

HKD = 3;

DHD = 4;

ALM = 5;

}

required Type type = 1;

enum Operate

{

SIGNIN = 1;

SIGNOUT = 2;

}

required Operate operate = 2;

required string cid = 3;

optional string cname = 4;

}

message ComponentResponse

{

required int32 result = 1;

}

#### 注册请求

* type：按照实际运行的组件实例类型设置；
* operate：SIGNIN；
* cid：组件ID标识，用CPU@进程ID标识；
* cname：组件名称。

#### 注册应答

* result：错误码

#### 注销请求

* type：按照实际运行的组件实例类型设置；
* operate：SIGNOUT；
* cid：组件ID标识，用CPU@进程ID标识；
* cname：组件名称。

#### 注销应答

无

#### 心跳请求

* type：按照实际运行的组件实例类型设置；
* operate：SIGNIIN；
* cid：组件ID标识，用CPU@进程ID标识；
* cname：组件名称。

#### 心跳应答

无

## 用户类

#### 协议定义

syntax = "proto2";

package msg;

message Author

{

required bool enableconfigure = 1;

required bool enablequery = 2;

}

message User

{

required string uname = 1;

required string password = 2;

required Author author = 3;

}

message UserRequest

{

enum Operate

{

NEW = 0;

DELETE = 1;

MODIFY = 2;

QUERY = 3;

LOGIN = 4；

}

required Operate operate = 1;

optional User user = 2;

optional string uid = 3;

}

message UserResponse

{

required int32 result = 1;

optional string uid = 2;

repeated User users = 3;

}

#### 新增请求

* operate：NEW；
* user.uname：用户名；
* user.password：用户密码；
* user.author.enableconfigure：配置权限许可标识；
* user.author.enablequery：查询权限许可标识。

#### 新增应答

* result：错误码；
* uid：用户ID标识。

#### 删除请求

* operate：DELETE；
* uid：用户ID标识。

#### 删除应答

* result：错误码。

#### 修改请求

* operate：MODIFY；
* uid：用户ID标识；
* user.uname：用户名；
* user.password：用户密码；
* user.author.enableconfigure：配置权限许可标识；
* user.author.enablequery：查询权限许可标识。

#### 修改应答

* result：错误码。

#### 查询请求

* operate：MODIFY；
* uid：用户ID标识。

#### 查询应答

* result：错误码；
* users：用户信息集合。

#### 登录请求

* operate：LOGIN；
* uid：用户ID标识；
* user.uname：用户名；
* user.password：用户密码。

#### 登录应答

* result：错误码；

## 设备类

#### 协议定义

syntax = "proto2";

package msg;

message Camera

{

required string cid = 1;

required int32 index = 2;

required bool enable = 3;

optional string nickname = 4;

optional string ip = 5;

}

message Device

{

required bool enable = 1 [default = false];

enum Factory

{

HIKVISION = 1;

DAHUA = 2;

EATON = 3;

}

optional Factory factory = 2 [default = NONE];

enum Type

{

DVR = 1;

NVR = 2;

IPC = 3;

}

optional Type type = 3;

optional string name = 4;

optional string password = 5;

optional string ip = 6;

optional int32 port = 7;

optional string did = 8;

optional string nickname = 9;

repeated Camera cameras = 10;

}

message DeviceRequest

{

enum Operate

{

NEW = 0;

DELETE = 1;

MODIFY = 2;

QUERY = 3;

}

required Operate operate = 1;

optional Device device = 2;

}

message DeviceResponse

{

required int32 result = 1;

repeated Device devices = 2;

}

#### 新增请求

* operate：NEW;
* device.enable：true表示启动设备，false表示停用设备；
* device.factory：设备厂商类型；
* device.type：设备类型；
* name：设备登录名；
* password：设备登录密码；
* ip：设备IP地址；
* port：设备端口号；
* nickname：设备别名。

#### 新增应答

* result：错误码；
* device.did：设备ID标识；
* device.cameras：设备可用摄像机集合。

#### 删除请求

* operate：DELETE;
* did：设备ID标识。

#### 删除应答

* result：错误码。

#### 修改请求

* operate：MODIFY;
* device.enable：true表示启动设备，false表示停用设备；
* device.factory：设备厂商类型；
* device.type：设备类型；
* name：设备登录名；
* password：设备登录密码；
* ip：设备IP地址；
* port：设备端口号；
* nickname：设备别名；
* device.cameras：设备可用摄像机集合。

#### 修改应答

* result：错误码。

#### 查询请求

* operate：QUERY;

#### 查询应答

* result：错误码；
* device：设备集合。

## 算法类

#### 协议定义

syntax = "proto2";

package msg;

message Algorithm

{

required string cid = 1;

optional int32 gpu = 2;

optional bool enablesleep = 3;

optional bool enablefight = 4;

optional bool enablehelmet = 5;

optional bool enablephone = 6;

optional bool enableattendancein = 7;

optional bool enableattendanceout = 8;

optional float similar = 9;

}

message AlgorithmRequest

{

enum Operate

{

CONFIGURE = 1;

QUERY = 2;

}

required Operate operate = 1;

repeated Algorithm algorithms = 2;

}

#### 配置请求

* operate：CONFIGURE；
* algorithm：算法参数集合。

#### 配置应答

无

#### 查询请求

* operate：QUERY。

#### 查询应答

* result：错误码；
* algorithms：算法配置集合。

## 员工类

#### 协议定义

syntax = "proto2";

package msg;

message Crew

{

optional string uid = 1;

optional string name = 2;

optional string job = 3;

repeated string pictures = 4;

}

message CrewRequest

{

enum Operate

{

NEW = 0;

DELETE = 1;

MODIFY = 2;

QUERY = 3;

}

required Operate operate = 1;

optional Crew crew = 2;

optional string uid = 3;

}

message CrewResponse

{

required int32 result = 1;

optional string uid = 2;

repeated Crew crews = 3;

}

#### 新增请求

* operate：NEW；
* crew：员工信息。

#### 新增应答

* result：错误码；
* uid：用户ID标识。

#### 删除请求

* operate：DELETE；
* uid：用户ID标识。

#### 删除应答

* result：错误码。

#### 修改请求

* operate：MODIFY；
* crew：员工信息。

#### 修改应答

* result：错误码。

#### 查询请求

* operate：QUERY。

#### 查询应答

* result：错误码；
* crews：员工信息集合。

## 报警类

#### 协议定义

syntax = "proto2";

package msg;

message Position

{

required int32 x = 1;

required int32 y = 2;

required int32 w = 3;

required int32 h = 4;

}

message Alarm

{

enum Type

{

NONE = 0;

SLEEP = 1;

FIGHT = 2;

HELMET = 3;

PHONE = 4;

FACE = 5;

}

required Type type = 1 [default = NONE];

repeated Position posistion = 2;

optional int32 uid = 3;

}

message AlarmMsg

{

required string cid = 1;

required string time = 2;

required string picture = 3;

required Alarm alarm = 4;

}

#### 报警上传

* cid：摄像机ID标识；
* time：报警时间；
* picture：报警实时图片，以JEPG格式封装；
* alarm：报警信息。

## 状态类

#### 协议定义

syntax = "proto2";

package msg;

message StatusRequest

{

enum Operate

{

SET = 1;

QUERY = 2;

}

required Operate operate = 1;

enum Type

{

SAIL = 1;

}

required Type type = 2;

optional bool flag = 3;

}

message StatusResponse

{

required int32 result = 1;

optional bool flag = 2;

}

#### 航停请求

* operate：SET；
* type：SAIL；
* flag：true表示航行，false表示停港。

#### 航停应答

* result：错误码。

#### 查询请求

* operate：QUERY。

#### 查询应答

* result：错误码。
* flag：true表示航行，false表示停港。