服务器技术文档

创建时间：2016-10-20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **修改时间** | **修改人** | **审批人** | **备注** |
| **V0.0.1** | 2016-10-20 | 张亚磊 |  | 创建文档 |

目录

[I. 服务器结构 3](#_Toc310409005)

[1. 服务器架构图 5](#_Toc310409006)

[2. Node线程模型](#_Toc310409007) 5

[II. 服务器环境部署](#_Toc310409008) 9

[III. 服务器功能](#_Toc310409013) 9

[1. 消息包结构](#_Toc310409014) 10

[2. 进程功能详解 10](#_Toc310409016)

[3. JS脚本模块 1](#_Toc310409020)2

[4. 客户端登录流程](#_Toc310409020) 13

[5. 配置管理](#_Toc310409020) 13

[6. 数据库操作接口](#_Toc310409020) 20

# 服务器结构

服务器是异步非阻塞的多进程多线程架构，运行于CentOS7操作系统。服务器底层使用C++编写，高效稳定，使用TCP协议进行进程间以及与客户端的通信，同时也支持、Websocket、Http协议。网络通信协议使用自定义格式，具备加密功能，安全高效。游戏数据库支持MongoDB和Mysql，可以选择自己想用的数据库。逻辑层使用简单高效的Javascript进行开发，集成Google V8引擎解析脚本，保证脚本运行效率，有js经验的开发人员可以快速上手。

服务器分为DaemonServer，MasterServer，CenterServer，GateServer，GameServer，PublicServer，ChatServer，DataServer，DBServer，LogServer，CrossServer。

DaemonServer：守护服务器，管理本机所有服务器进程的创建，重启，通过进程id控制，动态创建进程。

MasterServer：管理服务器，管理进程的状态，然后将进程状态同步给CenterServer管理，可以动态增加进程。

CenterServer：中心服务器，管理gate列表，选择gate，验证客户端token，管理game列表，选择game，管理所有在线玩家信息。

GateServer：网关服务器，维持客户端连接，消息转发，保存客户端sid与game的映射表

GameServer：游戏服务器，处理玩家游戏逻辑，场景，AOI等功能。

PublicServer：公共服务器，处理公会，排行榜，拍卖行等公共游戏逻辑。

ChatServer：聊天服务器，处理聊天，开房间，发红包等功能。

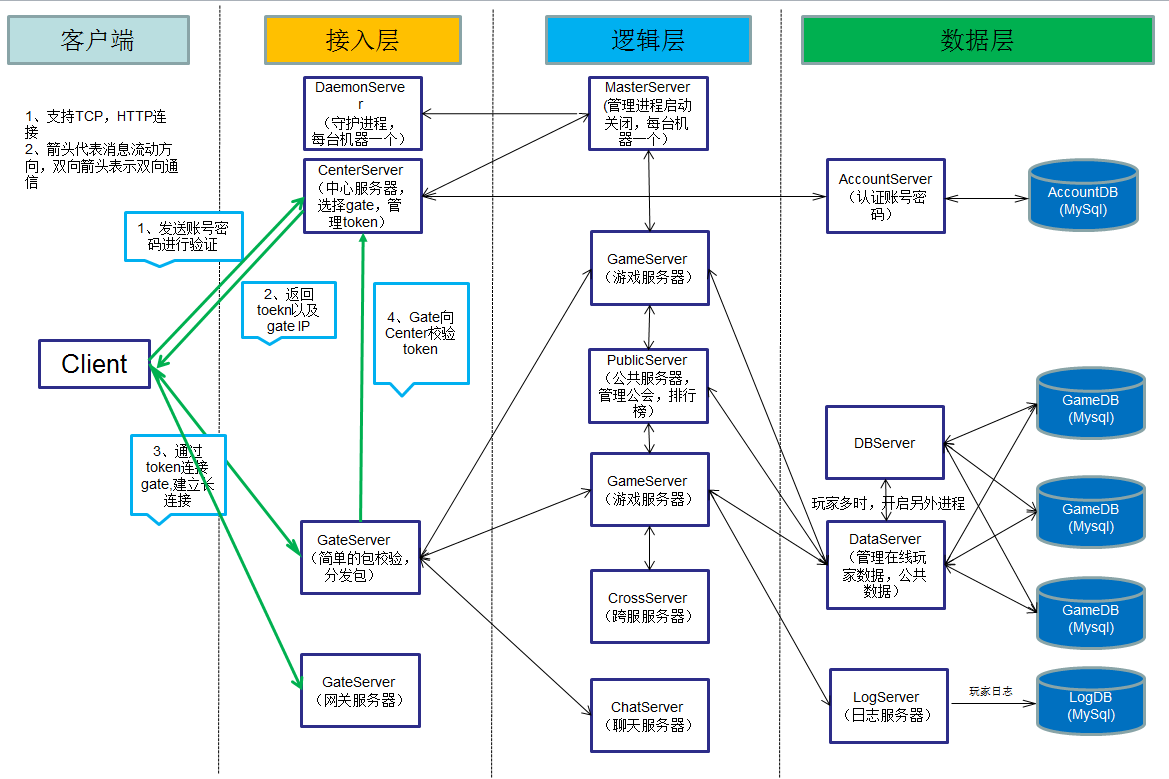
DataServer：数据服务器，提供在线玩家数据缓存和公共数据缓存，存取数据库，提供DBServer的路由算法。

DBServer：数据库服务器，数据缓存，存取数据。

LogServer：日志服务器，游戏日志存储功能。

CrossServer：跨服服务器。

1、**服务器架构图**



2、Node**线程模型**

**Server线程：**

此线程包括2个网络线程，表示接受连接的服务端。A\_B\_Server即为在A进程运行的供B进程连接的服务器模块，具体如下图

A\_B\_Server

Server\_Network

Server\_Accept

其中Server\_Accept线程负责监听端口，并接受客户端的连接封装成svc端点；Server\_Network线程负责从socket接受缓存中获取tcp包并解包，将解出后得到的完整逻辑包丢到服务器逻辑层以待逻辑层处理；负责将队列中待发送的数据包通过socket发送。

**Connector线程：**

此线程包括2个网络线程，表示发起连接的客户端。A\_Connector即为在本地进程向A进程发起连接的子线程组。连接建立通过该组子线程完成与服务端A之间的通信。具体如下图

A\_Connector

Connector\_Network

Connect\_Network负责向目标端口发起连接请求。并将对端封装成svc节点，然后从socket接受缓存中获取tcp包并解包，将解出后得到的完整逻辑包丢到服务器逻辑层以待逻辑层处理；负责将队列中待发送的数据包通过socket发送。

**Node\_Timer线程：**

定时器线程，用来处理定时器逻辑。

**V8线程：**

V8线程启动时候会加载主脚本文件，然后运行js脚本逻辑。

**Node\_Manager线程：**

进程内主线程，用来管理进程数据，网络线程等。

**Center\_Server：**

Timer

Center\_Client\_Server

V8\_Manager

Center\_Gate\_Server

**Gate\_Server：**

Gate\_Client\_Server

Gate\_Center\_Connector

Timer

V8\_Manager

Gate\_Node\_Server

**Game\_Server：**

Game\_Public\_Connector

Game\_Data\_Connector

Game\_Log\_Connector

Timer

Game\_Gate\_Connector

Game\_Gate\_Connector

Game\_Gate\_Connector

Game\_Master\_Connector

V8\_Manager

**Public\_Server：**

Public\_Log\_Connector

Public\_Server

Timer

Public\_Data\_Connector

V8\_Manager

Public\_Master\_Connector

**Log\_Server：**

Timer

Log\_Server

V8\_Manager

**Data\_Server：**

Timer

Data\_Server

V8\_Manager

**DB\_Server：**

Timer

DB\_Data\_Connector

V8\_Manager

**Center\_Server：**

Center\_Client\_Server

Timer

Center\_Node\_Server

V8\_Manager

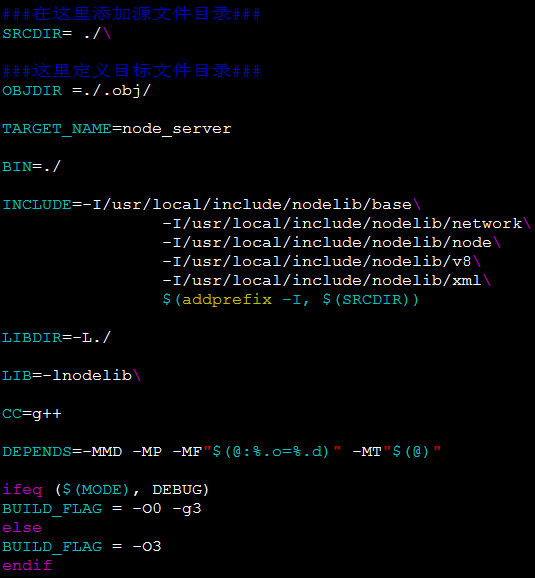
1. **服务器环境部署**

环境：安装Vm 10.0以上版本，Centos 7.0以上版本

运行环境：

使用root账号进入系统，将node\_env.tar.gz文件拷贝到系统中，运行命令tar –zvxf node\_env.tar.gz，解压，运行cd node\_env进入该目录，运行命令sh install\_server.sh，执行环境安装脚本。脚本会自动安装数据库和各种运行所需库。安装完成后会在/root下生成一个server目录，该目录下是安装好的服务器可执行文件以及资源文件。

一键部署脚本会自动安装所需的所有依赖。开发环境下还会拷贝静态库和头文件，修改代码后直接make就可以生成可执行文件。Makefile已经提供。如下：



如果源代码添加了新目录，请在SRCDIR中添加。INCLUDE和LIB是头文件目录和链接库路径，一般不需要改，如果链接了其他库就需要添加在这里。

node\_server内置了解析消息结构为js脚本以供v8运行时调用的功能。生成的目标文件是js/struct.js和js/message.js这两个脚本。生成方法是运行daemon\_server -s

服务器运行的时候执行daemon\_server程序，会直接创建各个服务器进程，不要直接执行生成的可执行文件node\_server

1. **服务器功能**
2. **消息包结构**

+-----------------+----------------------+----------------------+

| head\_bytes| readable bytes | writeable bytes |

| | (CONTENT) | |

+-----------------+----------------------+----------------------+

0 read\_index(init\_offset) write\_index vector::size()

message head:

int32(eid);

int32(cid);

uint16(len);

uint8(msg\_id);

uint8(msg\_type);

int32(sid);

其中eid表示endpoint\_id端点id,cid表示链接id,是服务器收到数据后写入消息里面的，msg\_type和sid字段是node消息才有的字段

1. **进程功能详解**

Daemon\_Server

守护服务器，管理本机所有服务器进程的重启，通过进程id控制。

启动服务器请执行本程序，不要直接执行node\_server

Master\_Server

管理服务器，管理进程的启动关闭，记录进程的状态，然后将进程状态转给DaemonServer管理，可以动态增加进程。

Center\_Server

Center进程是客户端向服务器发起连接需要接触的第一个进程。它的主要作用是接收客户端的登录请求，并在账号数据库中查询角色信息。如果查找失败，则会通知客户端创建角色。查找成功则生成token并返回该key和gate服务器的地址与端口给客户端，同时断开客户端连接。

Gate\_Server

Gate进程为网关进程。负责服务器和客户端之间消息的转发，隔离开逻辑服务器与客户端，对客户端发来的包进行校验，保证安全性。Gate进程在一组服务器环境中有多个副本，用于负载均衡。每个gate进程均需要连接所有的game逻辑进程和master进程。

客户端从center服务器获得token和地址端口等信息后连接gate服务器。并发送token进行验证。Gate服务器校验失败的话断开与客户端的连接。校验成功后gate会向特定的game服务器请求登陆，登陆成功后将角色信息返回客户端，并生成角色的sid（整个服务器系统中玩家角色的唯一标识），开始进行客户端消息和逻辑进程消息的中转。

Game\_Server

Game进程是处理游戏逻辑的主要进程。Game进程包含一个V8线程，拥有一个V8虚拟机实例，并读取脚本目录下的game\_server.js来执行逻辑处理。Game进程启动的时候根据配置文件读取相关场景和地图，并向master服务器报告注册自己的状态。Gate进程发来登陆请求后便开始登陆流程，通过DB进程获取玩家数据库中的信息返回给客户端，并在内存中生成玩家对应的实体对象。接着开始执行脚本中的大循环，处理具体逻辑。

Public\_Server

Public进程整个服务器组中只有一个，Gate进程和Game进程都要连接它，用来保存全局性质的数据，如公会，排行榜等，并对各个game的情况做统计，比如某个game上开辟了哪些场景等等。Master也含有一个V8线程，拥有自己的V8虚拟机实例，可以通过脚本编写Master的具体逻辑。

Chat\_Server

聊天服务器，处理聊天功能，开房间，发红包等功能。

Data\_Server

数据库管理进程，管理在线玩家的缓存数据和公共数据的缓存，目前已经支持了Mysql和Mongodb两种数据库。玩家登陆时拉取玩家信息，并每隔一段时间接收game或master传来的保存命令保存数据。

Log\_Server

保存日志信息的进程，接收服务器组中各个组件进程的日志消息，插入mysql的日志数据库中。

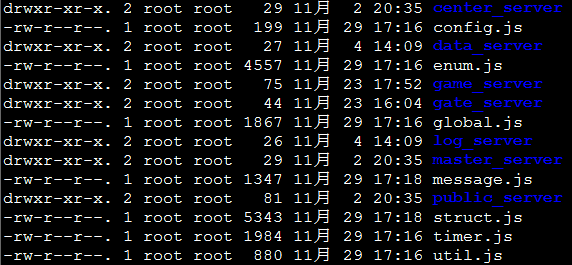
1. **JS脚本模块**

Javascript是服务器所使用的脚本语言。开发的过程中，程序员只需修改js脚本就能完成服务器功能的更新，不用重新编译C++代码。

服务器使用Google的v8引擎来解析脚本，每个节点进程中都分别运行了一份V8虚拟机实例。每个进程都有自己的xx\_server.js文件，该文件是是v8线程运行的主脚本，脚本开头将其他脚本加载进内存，并初始化一些全局变量，完成后底层逻辑大循环就会开始调用脚本里的接口。

Server.js脚本提供了5个供底层逻辑大循环调用的接口，分别是init, on\_drop, on\_msg, on\_tick,on\_hotupdate。init接口执行一些初始化操作，在脚本开始运行时被调用；on\_drop接口是连接掉线接口，当tcp层客户端连接断开时，on\_drop接口会被调用，参数是掉线客户端的cid（网络连接层对每个客户端的唯一标识）；on\_msg是消息处理接口，用来处理网络层收到的消息；on\_tick是执行定时操作的接口；on\_hotupdate是用来热更新配置文件的。

服务器脚本目录结构如下：



目录下的文件为目录名对应的进程运行所需的脚本文件。外层的js脚本文件是一些公共文件，提供对整个系统的支持。

**enum.js**：

该文件定义了程序需要的枚举类型，包括错误号，进程类型等。

**message.js、struct.js：**

message.js中定义了解析和发送消息时能使用的各种消息号，struct.js中则定义了包括消息体在内的所有js层需要使用的类型。

这两个文件都是自动生成的，不要手动更改其内容，运行daemon -s可以根据配置文件生成相应的脚本。需要更新的时候，要改相应的配置文件，并使用命令重新生成。

**config.js**

该文件是js读取配置文件功能，脚本初始化时便会根据config.js定义的全局变量和路径去读取供脚本层使用的json或xml配置文件，在开发过程中使用config来访问各种配置参数比较方便。

**util.js**

该文件定义了各种公共操作，比如取时间，判断类型等等。

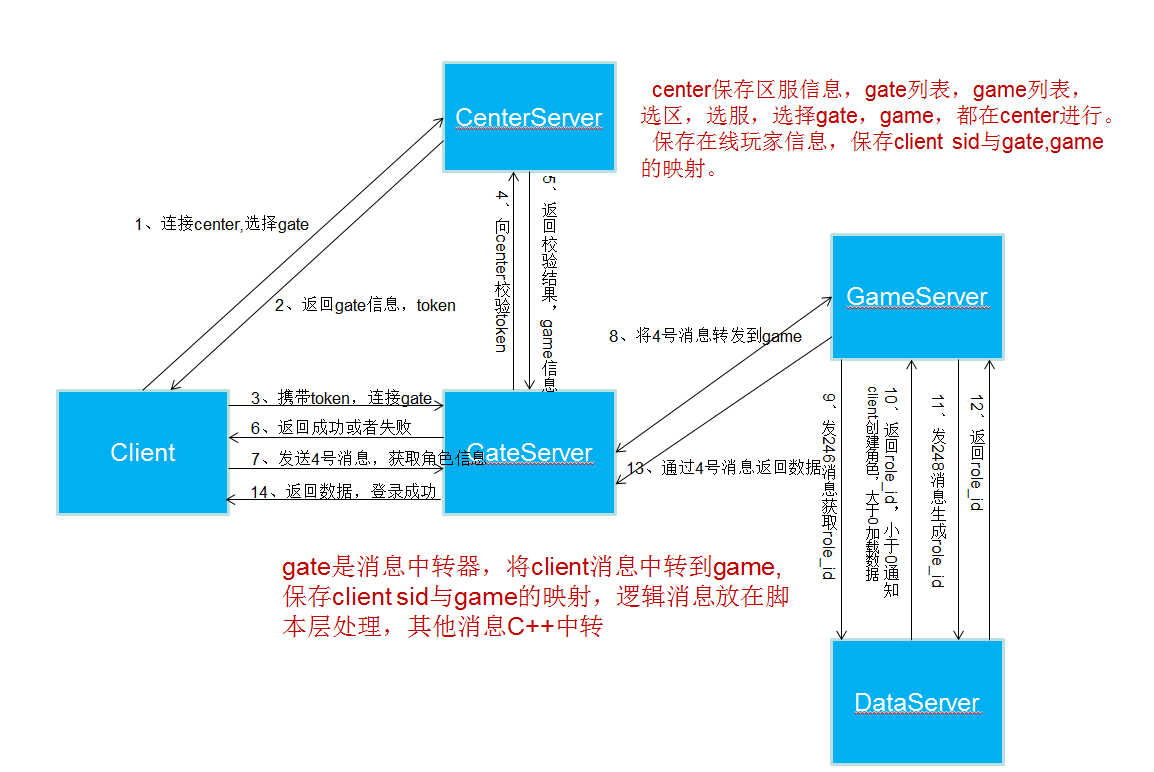
**timer.js**

该文件定义实现了定时器的各种操作。

**global.js**

该文件定义服务器使用的所有全局变量，全局变量定义在global类中，当做global的成员变量，不能直接定义。该文件不能热更新，也不能直接require。

1. **客户端登录流程**



增加session流程：gate增加session时候，先通知game，当game增加session完成后，发消息通知gate，此时gate再通知client建立session成功，可以保证正确的登陆顺序。

移除session流程：game移除session时候，先通知gate，当gate移除session完成后，发消息通知game，此时game再通知player下线，可以保证正确的下线顺序。另外当client直接断开与gate的网络层连接时候，gate需要先移除session,然后通知game让player下线。

1. **配置管理**

配置文件存放于根目录的config文件夹下。目前包含以下配置：



**sql**

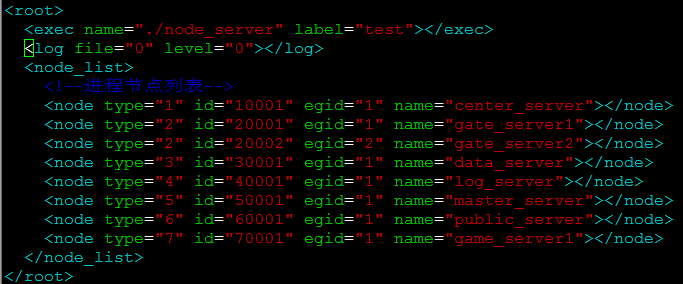
mysql脚本所在的目录，里面是mysql建立数据库所需脚本，部署环境的时候会自动安装数据库。当数据库结构发生变化时候，使用daemon –s命令自动生成sql脚本。

**node**

Node目录下存放的是服务器进程启动运行相关的配置文件。包含如下三种：



daemon\_conf.xml文件内容如图所示：

【exec】 name：表示 node\_server可执行文件的名称。

label：表示进程标签，用来区分进程属性。

【log】 file：表示是否将日志写入文件

Level：表示日志等级，小于该等级的日志不会被记录，日志等级依次为

Debug，Info，Warn，Error，Trace，Fatal，表示1,2,3,4,5,6

【node\_list】表示启动的node进程列表。每个进程表项包含以下4个参数：

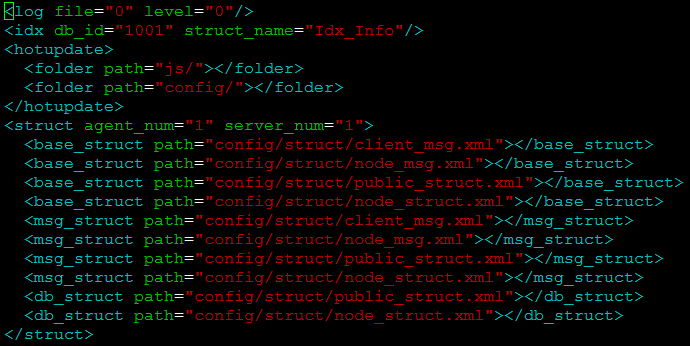
type：进程类型，和enum.js中的类型对应

id：进程的id标识

egid：网络端点的组id，有些进程属于同一类型，在node\_conf.json中同一类型的端点是定义在一起的，通过组id可以正常获取各进程对应的端点配置

name：进程的名称

node\_conf.xml如下所示：

 【log】 file：表示是否将日志写入文件

level：表示日志等级，小于该等级的日志不会被记录，日志等级依次为

Debug，Info，Warn，Error，Trace，Fatal，表示1,2,3,4,5,6

【idx】 db\_id：表示idx表所在id

struct\_name：idx结构体名称

【hotupdate】folder：需要热更新文件夹路径

【struct】log\_trace：打印日志开关，是否打印结构体读写日志

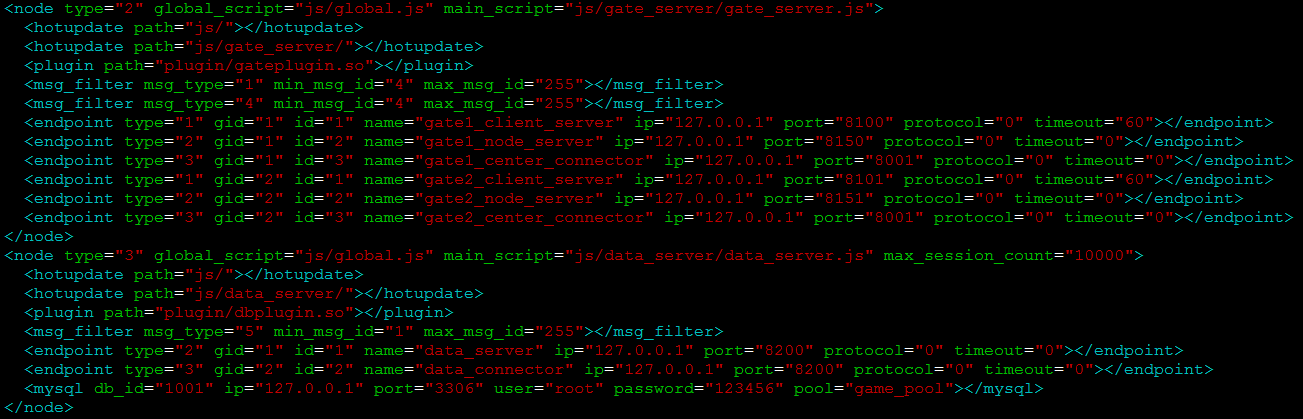
agent\_num：平台编号，生成角色id，帮派id时使用

server\_num：服务器编号，生成角色id，帮派id时使用

base\_struct：基础数据结构加载路径列表，表中是需要被加载为基础数据结构的配置表路径

msg\_struct：消息数据结构加载路径列表，表中是需要被加载为消息数据结构的配置表路径

db\_struct：需要保存到数据库中的DB数据结构加载路径列表，表中是需要被加载为DB数据结构的配置表路径

 【node】

type：进程的类型，跟enum.js中的类型号相对应

global\_script：全局数据定义的脚本路径，该脚本不能热更新。

main\_script：进程运行后加载的主脚本路径

hotupdate：需要热更新的文件夹路径

msg\_filter：消息过滤配置，

msg\_type：表示过滤的消息类型

min\_msg\_id：表示过滤的最小消息id

max\_msg\_id：表示过滤的最大消息id

plugin：需要加载的插件路径

endpoint：网络端点列表，配置包括server端和connector端的各种信息。

type：网络端点类型。1：接收客户端连接的server，2：接收服务器内部连接的server 3：发起连接的connector

gid：网络端点的组id。一个进程是一组端点，当该类型进程有多个实例时组id用以区分各个进程的端点信息。

id：该端点在当前组下的id

name：端点名称

ip：端点绑定的ipv4地址（connector组件时则是连接的目的地址）

port：端点绑定的端口号（connector组件时则是连接的目的端口）

protocol：协议类型。0：tcp 1：udp 2：http 3：websocket

timeout：服务器端点心跳超时时间。客户端连接的类型需要设置，内部组件一律为0（无超时时间）

msyql：表示msyql数据库配置

db\_id：mysql数据库的编号，一个编号标志着一个mysql服务。mysql编号范围为1000~1999。

ip：需要连接的mysql地址

port：需要连接的mysql端口

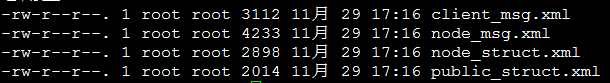
user：mysql用户名

password：mysql密码

pool：mysql连接池的名称

**struct**

该目录下有以下配置表：

 struct目录下的配置表都是xml格式，用来定义在游戏中用到的各种数据结构。服务器xml解析用的是开源工具tinyxml，很小巧又功能齐全的一个xml解析工具。配表的时候，整张表需要新建一个根节点，名称为“root”，在根节点内新增的子节点就是服务器能读取并使用的具体节点。

**标签说明**

标签名即是该数据结构能被其他xml所引用的类型名。每个结构中都有子标签。

head子标签是该结构的头信息，包含head子标签的结构有两种类型。一种是将来要存入数据库的数据类型，其中table\_name属性是保存入数据库时的表名，index\_name属性是保存的时候用来作为索引的变量名。另一种是消息类型，msg\_id属性是该消息的消息号，msg\_name属性是该消息能在js脚本中被引用的消息ID枚举名称。

arg标签表示该结构含有一个基础数据类型成员，type属性为具体基础类型，name属性为该成员在脚本中可被引用的字段名。

vector标签表示该结构有一个数组成员，type属性是数组成员的类型，name是可被引用的变量名。

map标签表示该结构有一个map对象， key\_name表示map的key字段名，key\_type表示key字段类型，key\_bit表示key字段位数，type表示为map里value的类型，bit表示value的位数，name表示map本身的字段名。

注意：vector和map在配置时候，需要配置vibt字段，表示长度字段位数

struct标签表示该结构含有一个复合结构字段，type是类型名，bit表示字段位数，name为可被引用的字段名。

if标签表示该结构体内子字段是否存在，type是类型名，bit表示字段位数，name为可被引用的字段名。If标签内子字段表示相应子字段的值。

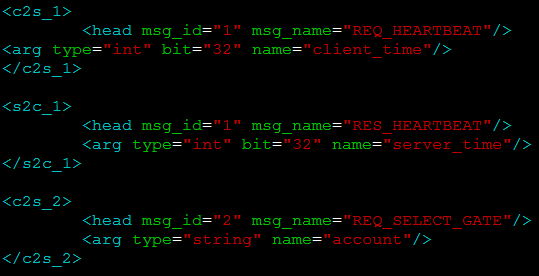
switch标签表示按照name字段的值去选择相应的case标签对应的子字段。

**数据类型说明**

现在支持的数据类型有：int,uint,int64,uint64,float,bool,string,其中int,uint类型在配置的时候，需要配置bit字段，用来表示数据位数，vector和map配置时候，需要配置vbit字段，表示数组长度

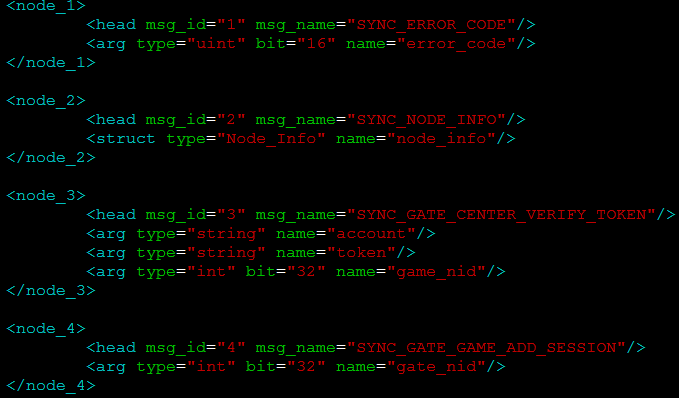
**client\_msg.xml**

客户端和服务器之间通信所用的基本数据结构。如下所示：



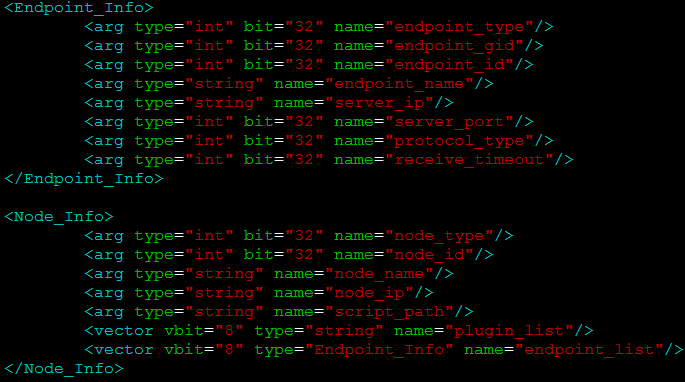
<c2s\_xxx>表示这是一个从客户端发往服务端的消息结构。Xxx是其消息号。标签名c2s\_xxx会被转化为js脚本中的function名称，可以直接new出来进行数据操作。Type属性可以引用其他xml文件中定义的数据结构名。<s2c\_xxx>则表示这是一个从服务端发往客户端的消息。命名规则同上。

**node\_msg.xml**



服务器间的内部消息结构。标签名是js中的function名。命名规则同client\_msg.xml。

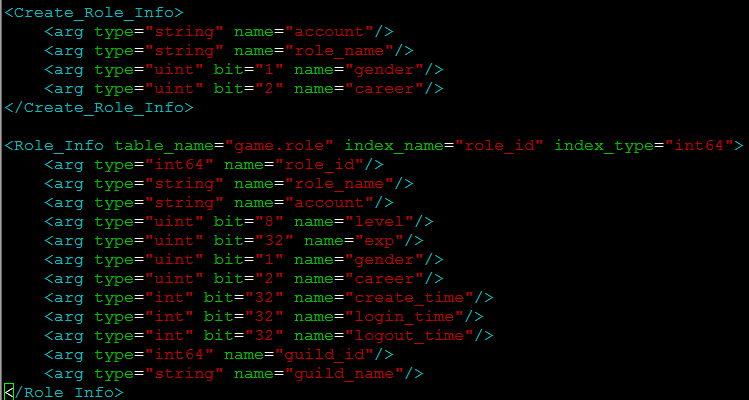
**node\_struct.xml**



服务器内部所使用的数据结构，包括需要保存到数据库的数据。可以被msg文件引用。

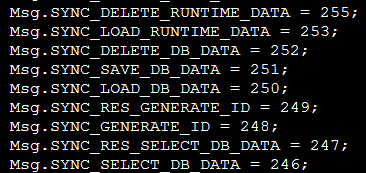
**public\_struct.xml**

客户端和服务端通用的数据结构。



1. **数据库操作接口**

数据库操作是通过向data\_server进程发node消息形式，总共有10个消息，消息内容定义在config/struct/node\_msg.xml中，内容如下：



【246】表示按照条件查询数据库指定值或者查询数据库一行记录，返回消息是247消息。

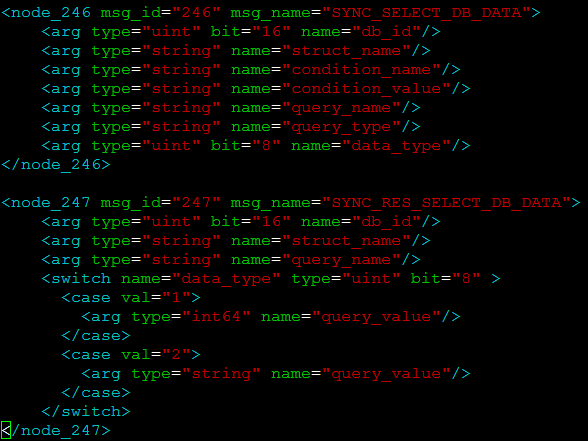
db\_id：表示操作的数据库id，

struct\_name：表示数据库表对应的结构体名称

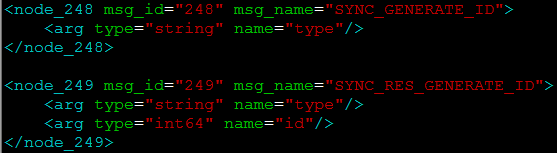
condition\_name和condition\_value代表查询条件

query\_name和query\_type表示查询的字段名称和类型，可以为空，为空时候表示查询一行的值

data\_type：表示查询的逻辑类型，与247号消息的case标签对应



【248】表示生成指定类型的id，该id是64位整数，全服唯一，返回的是249号消息



【250】按照int64类型的key\_index从数据库加载数据，当key\_index为0时候表示加载整张表数据，返回的是251号消息。

【251】保存数据到数据库。

save\_type：保存类型，1表示保存到缓存，2表示保存到数据库和缓存，3表示保存到数据库并清空缓存

vector\_data：表示保存的数据是否为数组

db\_id：表示操作的数据库id

struct\_name：表示数据库表对应的结构体名称

data\_type：保存数据类型

【252】删除数据库数据

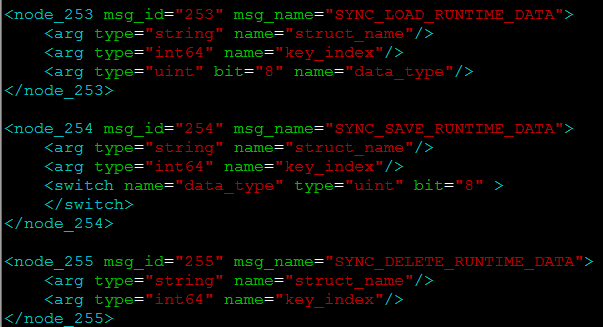
index\_list：要删除的数据库索引列表



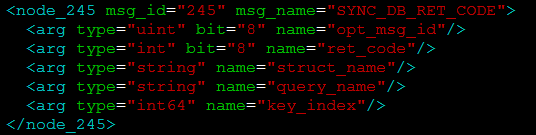
【253】按照int64类型的key\_index加载运行时数据，返回的是254号消息

【254】按照int64类型的key\_index保存运行时数据

【255】按照int64类型的key\_index删除运行时数据

**

**注意：所有的数据库操作在出现错误时候，都会返回245号消息，返回相应的信息，供逻辑层判断，消息定义如下：**

 **opt\_msg\_id：请求数据操作的消息id**

**ret\_code：返回代码，0表示成功，-1表示操作失败，-2表示数据不存在**

**struct\_name：操作的结构体名称**

**query\_name：查询的字段名称**

**key\_index：操作的数据库索引**