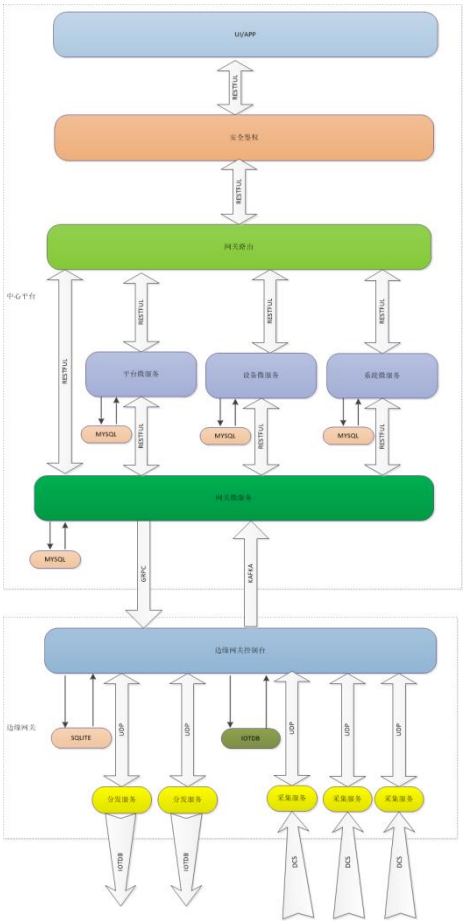
边缘网关系统设计说明书

1. 系统架构

系统结构在边缘侧采用服务管理机制，边缘网关同服务间采用 UDP 连接进行通讯。基本原则时尽量轻量化服务的设计，不引入第三方组件。

系统架构如下：



2. 边缘网关组件

边缘网关组件包括：

1）主控

负责接收 GRPC 消息，协调和管理内部各组件，内部采用线程启动每个组件，定时扫描组件状态，如果组件不正常，需要重新拉起组件。

2）告警模块

负责记录各组件以及服务告警，将告警通过 KAFKA 上报给中心侧，告警码需要规范统一，并且能够过滤告警，防止告警抖动，重复上报。

3）日志模块

负责收集边缘网关、采集服务和分发服务等服务日志，可以按照指定的时间范围收集，收集好的日志可以打包，并通过 GRPC 返回给中心侧。

4）性能统计模块

性能统计模块负责统计数据库性能，系统运行性能等，将统计的数据通过 KAFKA 定时上报给中心侧。

5）流量监控模块

主要负责监控测点流量数据，包括测点流量趋势，测点接收的数据等。将收集到的按照规定进行分析整理，通过 KAFKA 定时上报给中心侧。

6）采集服务管理模块

负责管理采集服务，接收并处理采集服务获取的数据，将获取的数据进行标准化，调用算法模块进行预处理，将处理后的结果存入到数据管理模块中。

7）分发服务管理模块

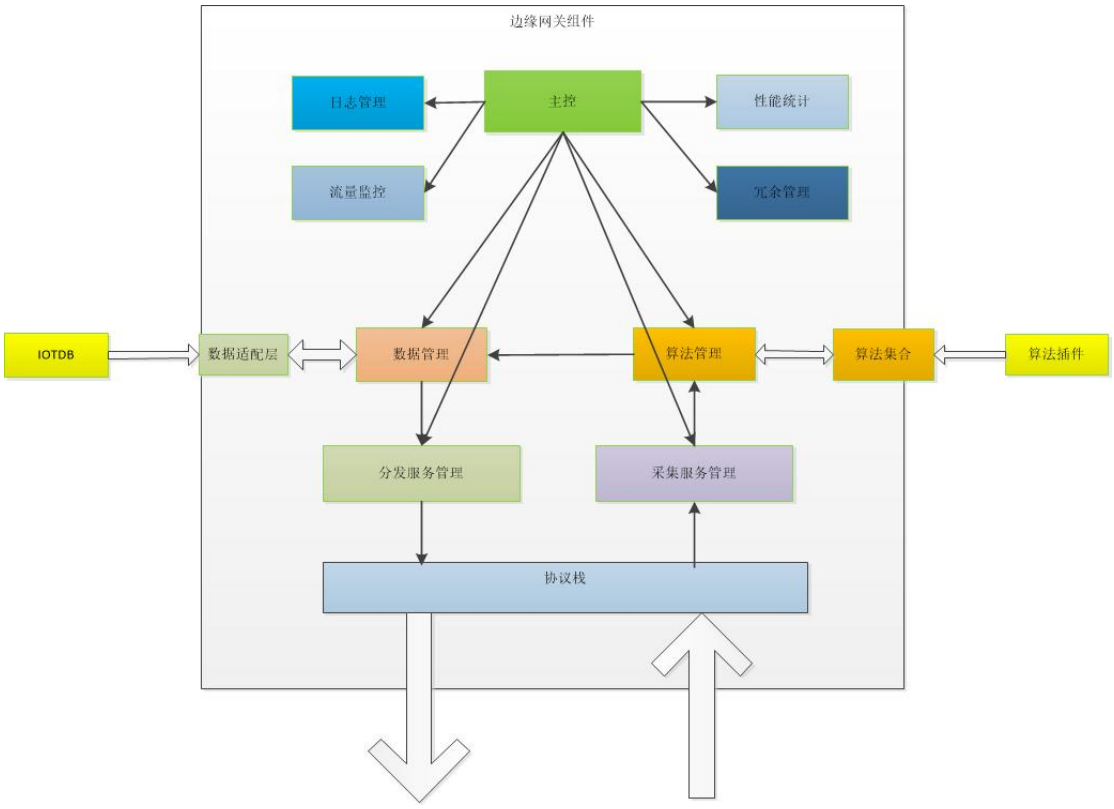
负责管理分发服务，接收外部订阅，检测数据变化，将变化的数据提交给分发服务，支持分发给多个服务。

8）算法管理模块

算法管理模块主要是接收界面下发的算法库，并存入到算法集合，处理采集服务提交的数据，通过配置的算法进行预处理。

9）数据管理模块

数据管理模块为边缘侧的数据中心，将采集的数据进行缓存，并通过调用数据适配器持久化到 IOTDB 中，分发服务通过订阅获取到对应的测点数据。

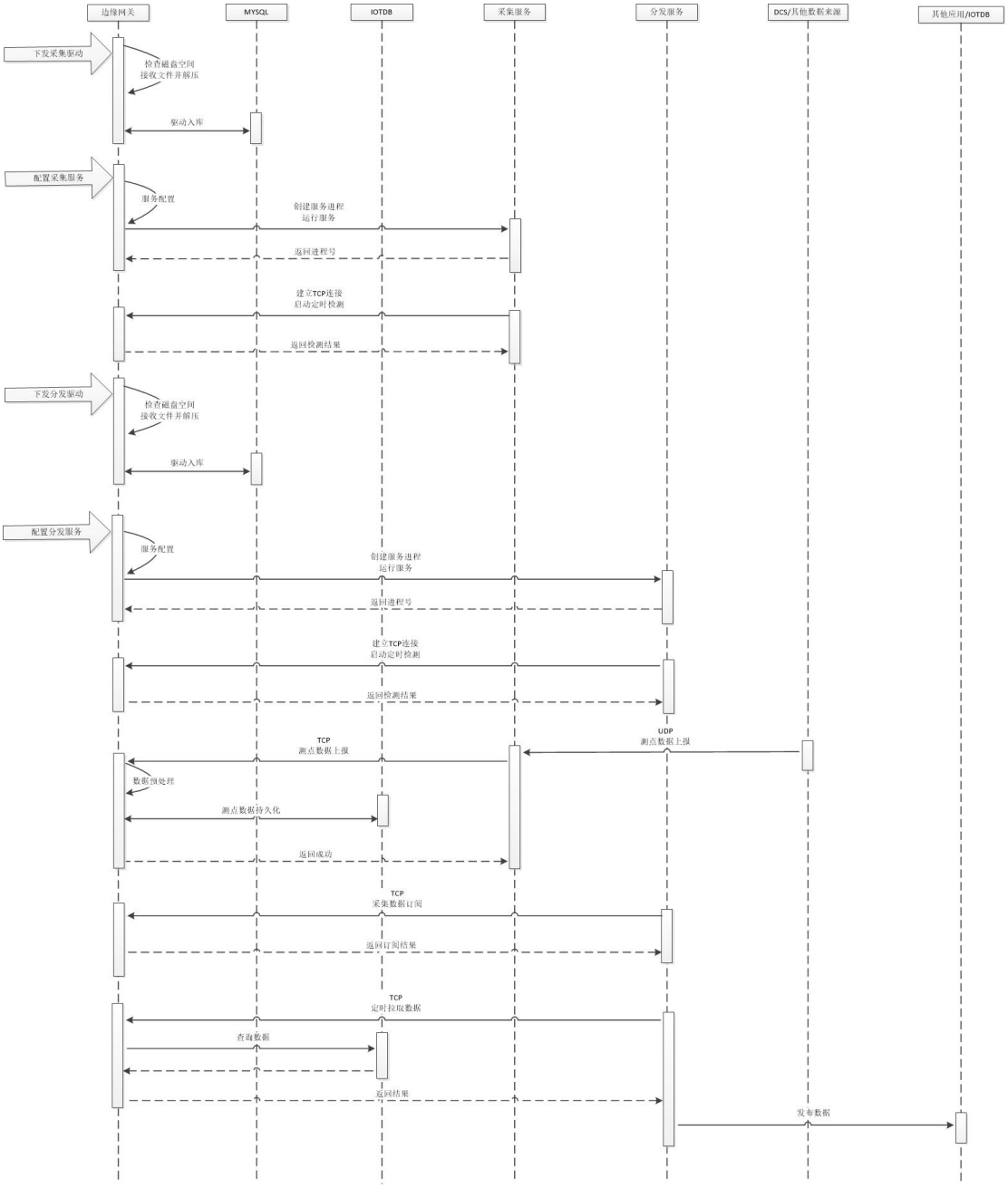


3. 系统设计

边缘网关的作用是管理各种服务，与中心侧对接，将驱动及服务配置信息下发到对应服务器并进行控制和监管，配置测点信息等。

边缘网关同中心侧网关配置信息采用 GRPC 连接，边缘网关作为 GRPC 服务端，边缘网关 GRPC 地址和端口通过 KAFKA 上报给中心侧。

边缘网关和服务之间采用 TCP 连接，边缘网关作为 TCP 服务端，各服务作为客户端接入。采集服务在启动时，需要根据配置文件连接边缘网关。并定时上报状态和数据。分发服务在启动时，也需要根据配置文件连接边缘网关，并订阅测点数据。



4. 边缘网关

边缘网关主要作用是接收中心侧配置信息，对驱动进行管理，根据驱动配置和启动服务，下发测点信息等，根据服务获取到的数据进行汇总，分析，转发和上报。

4.1 边缘网关注册

边缘网关采用 KAFKA 进行数据上报，上报内容包括边缘网关的 IP 和 GRPC 端口，对应的 KAFKA 接口如下：

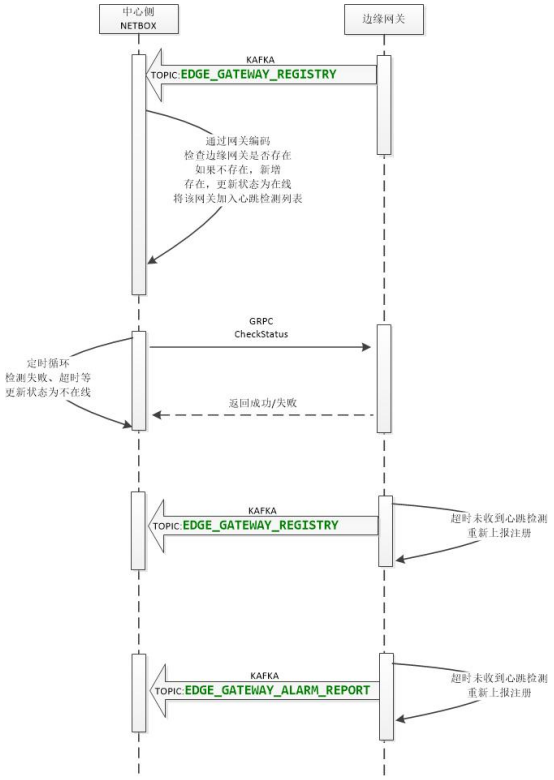
TOPIC:**EDGE_GATEWAY_REGISTRY**

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message EdgeGatewayInfo {
  required string ServiceName=1; //边缘网关名称，用于在主动上报时，区分边缘网关
  required string ServiceCode=2; //网关编码
  required int32  Type=3;        //网关类型
  required string Ip=4;          //IP 地址
  required int32  Port=5;        //GRPC 连接端口
  required int32  FilePort=6;    //文件传输连接端口
  required int32  Status=7;      //边缘网关状态
  required string Version=8;      //边缘网关版本
  required string Remark=9;      //备注
  optional string Additional=10;    //附加信息
}
```

4.2 边缘网关心跳检测

边缘网关在启动或超时未收到心跳检测时，需要通过 KAFKA 主动发起注册机制，注册后，中心侧会定时进行心跳检测，以保证边缘网关运行正常，并且未脱管。

4.2.1 流程设计



4.2.2 接口设计

GRPC 接口如下：

//网关健康检查

```
rpc checkStatus(EdgeInfo) returns(ErrCode) {}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

message EdgeInfo {
  required string EdgeName=1;          //网关名称
  required int32  Type=2;              //网关类型
  required int32  Status=3;            //状态
  required int32  Count=4;             //检查次数
  required string Remark=5;           //备注
  optional string Additional=6;         //附加信息
}
```

返回通用错误信息如下

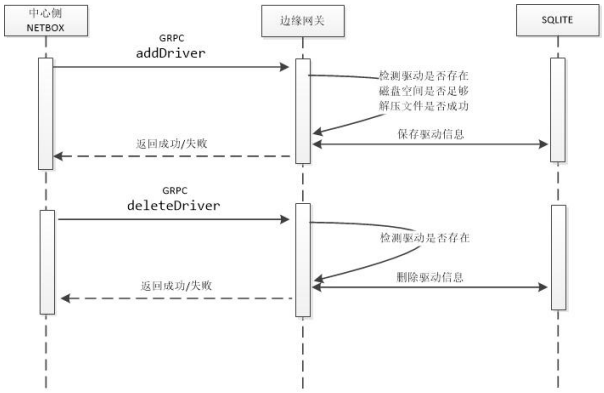
```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

//错误编码
message ErrCode {
  required int32 ID     = 1; //编号
  required string PN    = 2; //名称/标题
  required int32 Number = 3; //错误码
  required string Flag  = 4; //描述
}
```

4.3 驱动下发

驱动下发包括新增驱动和删除驱动，驱动是通过驱动名称和版本号确定唯一驱动，同一个驱动名称和版本，只能下发一次，驱动下发后，驱动目前只支持 TAR.GZ 格式和 ZIP 格式，每个驱动中需要有 config.json 文件，该文件为 JSON 格式，内容为每个驱动在配置服务时需要的参数，参考格式如下：

```
{
    "name":"dcs_recv_driver",
        "platform":"WIN32",
        "version":"v5",
        "description":"TEST DRIVER",
    "channel": [
    ],
    "labelWidth": "100px",
    "other": [
    ],
    "server": [
        {
            "cascadeRules": [
            ],
            "desc": "",
            "extend": {
            },
            "h": 2,
            "i": "0",
            "id": "nt8o1cb6bcw0000000",
            "key": "log_cache_mode",
            "moved": false,
            "options": [
                {
                    "label": "日志服务",
                    "value": "outer"
                },
                {
                    "label": "内部缓存",
                    "value": "inner"
                }
            ],
            "props": {
            },
            "scope": "advanced",
            "title": "日志缓存模式",
            "type": "select",
            "value": "inner",
            "w": 12,
            "x": 0,
            "y": 2
        },
        {
            "cascadeRules": [
                {
                    "key": "log_cache_mode",
                    "orAnd": "",
                    "symbol": "==",
                    "value": "outer"
                }
            ],
            "desc": "日志服务 IP 地址",
            "extend": {
            },
            "h": 2,
            "i": "1",
            "id": "sjxozg9lse80000000",
            "key": "log_collection_addr"
```

4.3.1 流程设计



4.3.2 接口设计

GRPC 接口：

//新增驱动

```
rpc addDriver(DriverInfo) returns(ErrCode) {}
```

//删除驱动

```
rpc deleteDriver(DriverInfo) returns(ErrCode) {}
```

消息内容如下：

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

message DriverInfo {
  required string DriverName=1;       //驱动名称
  required int32  DriverType=2;       //驱动类型
  required string Version=3;          //驱动版本
  required string Ip=4;              //IP 地址
  required int32  Port=5;            //文件传输连接端口
  required int32  OperationType=6;    //操作类型 0-下发 1-删除
  required string FileName=7;         //文件名称，不包含路径，包含扩展名
  required string FileType=8;         //文件类型 0-ZIP 1-GZ 2-RAR 3-EXE
  required int64  FileSize=9;         //文件大小
  required string Remark=10;          //备注
  optional string Additional=11;      //附加信息
}
```

4.3.3 数据库设计

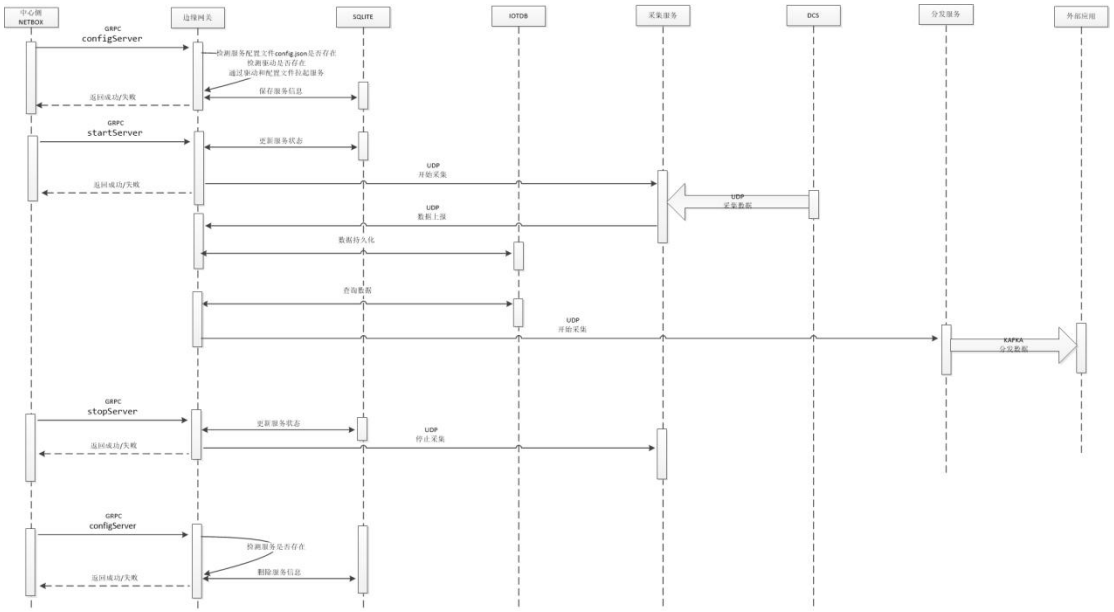
a) 驱动表(netbox_driver)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	driver_name	varchar(200)	驱动名称
3	driver_type	int(4)	驱动类型
5	version	varchar(30)	驱动版本
6	file_name	varchar(200)	文件名称
7	file_type	int(4)	文件类型
8	file_path	varchar(300)	文件保存路径
9	status	int(4)	使用状态：0-未使用，1-已使用
10	create_time	datetime	创建时间
11	remark	varchar(1000)	备注信息
12	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

4.4 服务配置

服务配置包括新增服务，启动服务，停止服务，服务删除和服务状态检测。新增服务是通过新增服务配置，运行驱动程序。

4.4.1 流程设计



4.4.2 接口设计

GRPC 接口：

//服务配置

```
rpc configServer(ServerInfo) returns(ErrCode) {}
```

//开启服务

```
rpc startServer(ServerInfo) returns(ErrCode) {}
```

//停止服务

```
rpc stopServer(ServerInfo) returns(ErrCode) {}
```

消息内容如下：

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

message ServerInfo {
  required string ServerName=1;       //服务名称
  required string ServerCode=2;       //服务编码
  required int32  ServerType=3;       //服务类型 0-采集 1-分发
  required string DriverName=4;       //所属驱动名称
  required string Version=5;          //所属驱动版本
  required int32  StartMode=6;        //启动模式
  required int32  ConfigType=7;       //配置类型
  required int32  LogType=8;          //日志缓存模式
  required int32  LogLevel=9;          //日志级别
  required string LogPath=10;          //日志路径
  required int32  serverMode=11;       //通信模式 1-MQTT 2-GRPC
  required int32  OperationType=12;    //操作类型 0-配置 1-运行 2-停止 3-删除
  required string Remark=13;           //备注
  optional string Additional=14;        //附加信息
}
```

4.4.3 数据库设计

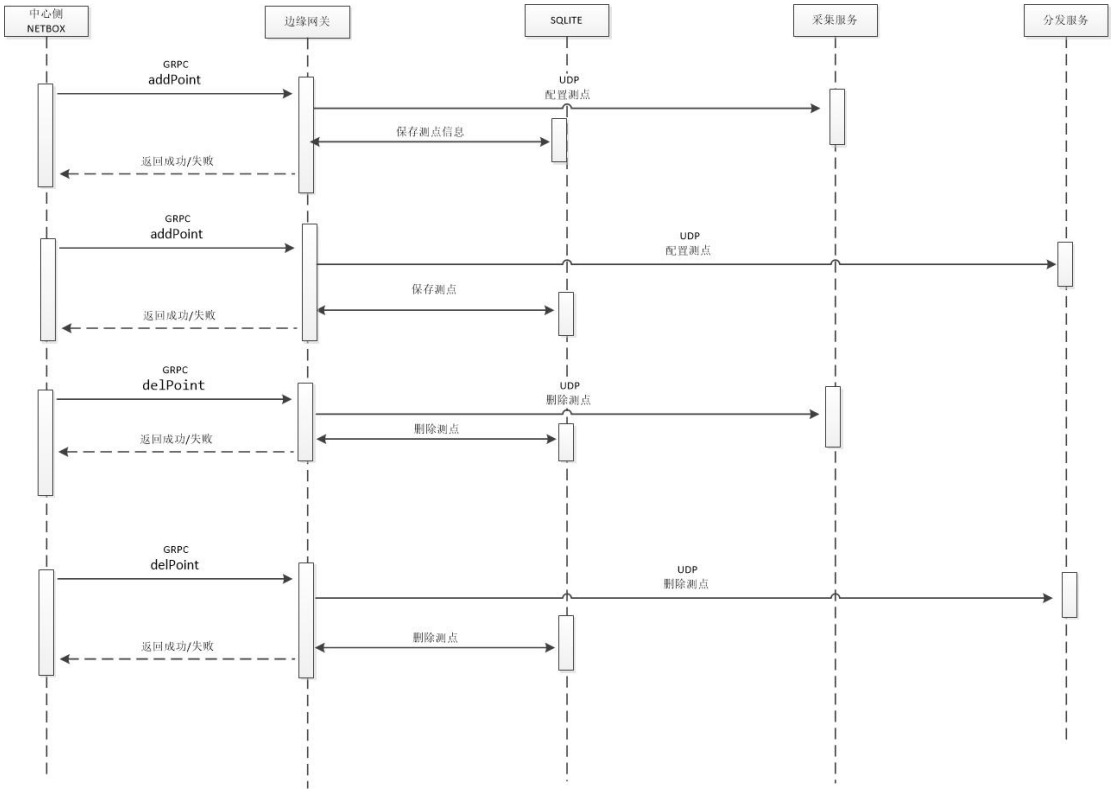
b) 服务表(netbox_server)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	server_name	varchar(200)	服务名称
3	server_code	varchar(200)	服务编码
4	server_type	int(4)	服务类型
5	status	int(4)	使用状态：0-未启动，1-已启动，2-停止
6	create_time	datetime	创建时间
7	driver_id	bigint(20)	驱动 ID，对应驱动表中的 id
8	remark	varchar(1000)	备注信息
9	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

4.5 测点配置

通过中心侧将需要监测的测点信息配置给对应的服务，采集服务对于同一个测点只允许配置一次，分发服务不做限制。

4.5.1 流程设计



4.5.2 接口设计

GRPC 接口：

//新增测点

```
rpc addPoint(PointInfo) returns(ErrCode) {}
```

//删除测点

```
rpc delPoint(PointInfo) returns(ErrCode) {}
```

消息内容如下：

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
import "pointattr.proto";
package edge;

message PointInfo {
  required string    TaskName=1;           //名称
  required string    TaskCode=2;           //编码
  required string    ServerName=3;         //所属服务名称
  required string    ServerCode=4;         //所属服务编码
  repeated PointAttr Points=5;             //测点信息
  required string    Remark=6;             //备注
  optional string    Additional=7;         //附加信息
}
```

4.5.3 数据库设计

c) 测点表(netbox_point)

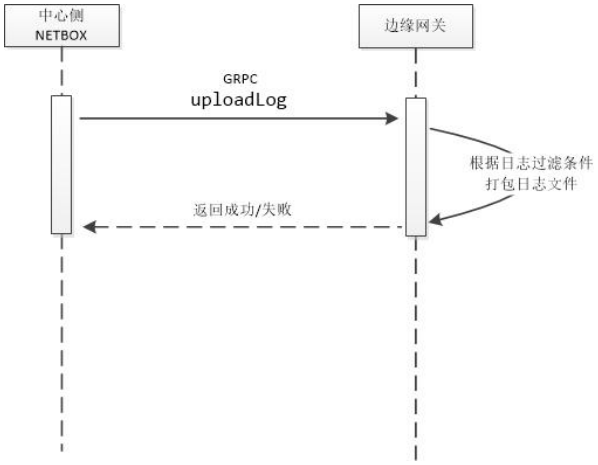
序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	point_name	varchar(200)	测点名称
3	point_code	varchar(200)	测点编码
4	point_type	int(4)	测点类型 0-boolean 1-int32 2-int64 3-float 4-double 5-text
5	max_value	double(8,2)	最大值
6	min_value	double(8,2)	最小值
7	attribute	varchar(60)	附加属性
8	status	int(4)	使用状态：0-未配置，1-已配置
9	create_time	datetime	创建时间
10	server_id	bigint(20)	服务 ID，对应服务表中的 id
11	remark	varchar(1000)	备注信息
12	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

4.6 日志管理

日志管理主要是用来采集边缘网关和服务的运行日志，通过在中心侧指定上传日志的时间范围，将范围内的日志打包压缩后上传到中心侧，并通过页面下载到本地查看。

日志管理可以根据查询条件，指定查询时间范围，查询边缘网关或是服务的日志。

4.6.1 流程设计



4.6.2 接口设计

//日志上传

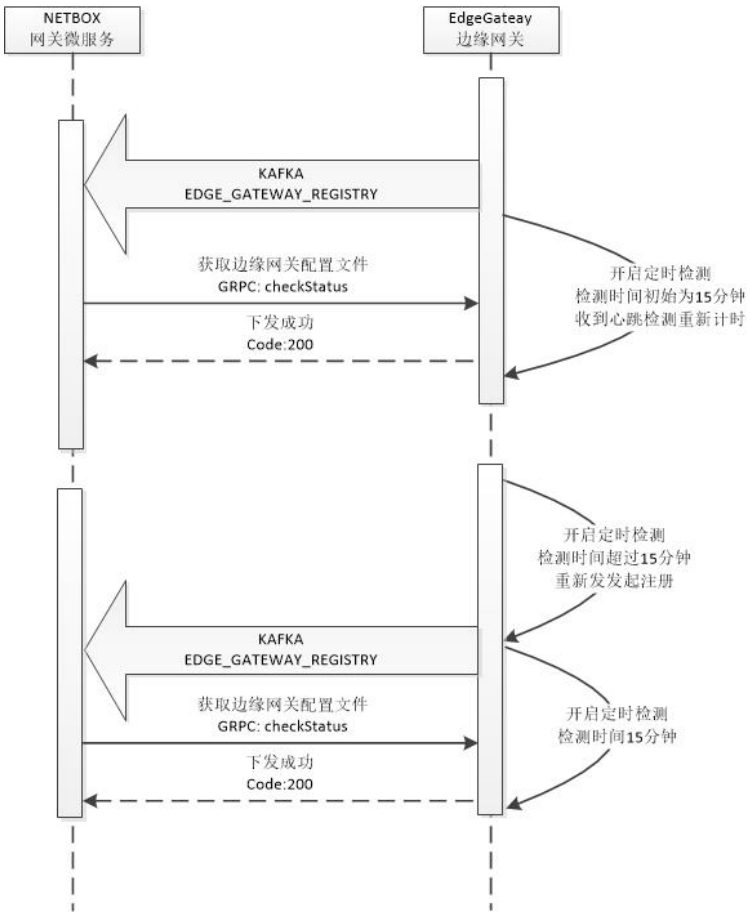
```
rpc uploadLog(LogFilter) returns(stream LogFile) {}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

message LogFilter {
  required int32  Type=1;               // 0-网关，1-服务
  optional string ModuleName=2;          //名称
  optional string ModuleCode=3;          //编码
  optional string StartTime=4;           //开始时间
  optional string EndTime=5;             //结束时间
  optional string Remark=6;              //备注
  optional string Additional=7;          //附加信息
}

message LogFile {
  required bytes buffer=1;      //文件流
}
```

4.7 告警管理

告警管理包括边缘网关超时未收到心跳检测，运行的服务异常或终止等，需要通过 KAFKA 上报告警，便于用户查看并快速解决告警。

4.7.1 流程设计



4.7.2 接口设计

KAFKA 消息上报告警：

TOPIC: **EDGE_GATEWAY_ALARM_REPORT**

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message AlarmInfo {
  required string uuid=1;              // 编号，唯一标识
  required string AlarmName=2;         // 告警名称 运行，新增，删除，启动，停止，退出等
  required int32 AlarmModule=3;        // 告警模块 0-网关，1-服务
  required string ModuleName=4;        // 模块名称
  required string ModuleCode=5;        // 模块编码
  required int32  AlarmType=6;         // 告警类型 0-提示，1-告警，2-异常，3-终止
  required int64  AlarmCode=7;         // 告警码
  required string AlarmMsg=8;          // 告警提示
  required string AlarmInfo=9;          // 告警信息
  required string AlarmSuggestion=10;   // 消除告警建议
  required int32 AlarmStauts=11;        // 告警状态 0-未处理， 1-已屏蔽，2-已消失，4-已处理
  required string AlarmTime=12;        // 告警时间
  required string Remark=13;           // 备注
  optional string Additional=14;       // 附加信息
}
```

5. 采集/分发服务

采集服务主要作用是进行数据格式转换及上报，能够轻量化运行。对外接口采用 UDP 连接，服务作为客户端，主动访问边缘网关，服务的接口采用标准化头。

分发服务主要作用是能够根据测点订阅边缘网关的数据。对外接口也采用 UDP 连接，服务作为服务端，订阅数据后等待边缘网关进行数据上报。

5.1 状态上报

服务需要定时上报自身状态信息给边缘网关，如果超时未搜到网关回复，需要记录告警。

5.2 测点配置

边缘网关会通过 UDP 对监测的测点信息进行增删。

5.3 采集测点数据上报

报文头采用 32 个字节表示，数据区使用 12 个字节循环，整体长度为 12*测点个数，单位为字节，如果为检测包，则测点个数字段为 1，数据区长度为 12 字节，4 个字节为记录检测区间长度，8 个字节为发送的测点包个数：

序号	字段名		类型	长度（字节）	说明
1	消息序号		无符号长整形	8	从 0 递增，达到最大值后从 0 循环
2	数据包类型		无符号短整形	2	0-未知，1-数据包，2-告警包，3-心跳包
3	域号		字节	2	数据发送的域编号
4	站号		字节	2	数据发送的站编号
5	时间（秒）		无符号整形	4	长整型数，精确到秒
6	时间（毫秒）		整型数	2	整型数，毫秒
8	分包数		字节	1	第几包，如果不分包填 0
9	分包号		字节	1	第几包，如果不分包填 0
10	检验码		无符号整形	4	测点数据校验码
11	包中测点个数		无符号短整形	2	测点包中的个数，不大于 1000
12	测点文件版本		无符号整形	4	点文件列表版本号
13	数据区	时间（秒）	无符号整形	4	
		时间（毫秒）	无符号短整型数	2	
		时间（微秒）	无符号短整型数	2	
		测点编号	无符号整形	4	
		类型	无符号短整型数	2	类型： 1-BOOLEAN（布尔值） 2-INT32（整型） 3-INT64（长整型） 4-FLOAT（单精度浮点数） 5-DOUBLE（双精度浮点数） 6-TEXT（字符串）

返回消息报文定义长度 32 字节：

序号	字段名	类型	长度（字节）	说明
1	消息序号	无符号长整形	8	从 0 递增，达到最大值后从 0 循环
2	检查结果	无符号短整形	4	0-成功，1-数据缺失，2-校验码不通

				过，3-消息头不正确，4-测点数据不正确
3	消息长度	字节	4	接收到消息长度
4	测点个数	字节	4	接收到测点个数
5	时间范围	字节	4	接收到的时间范围
6	数据包类型	无符号短整形	2	0-未知，1-数据包，2-告警包，3-检测包
7	分包序号	整型数	2	第几包，如果不分包填 0
8	检验码	无符号整形	4	消息内容校验，每个字节累加值

5.4 UDP 协议层

因数据采用 UDP 传输，边缘网关和采集/分发服务之间需要封装协议层，用来解决 UDP 传输的不可靠及分包排序问题。协议层的作用是让上层服务不感知数据传输的效率，排序以及重发机制。

1）数据传输采用快速传输，则不能采用同步机制，即每发送一个包，需要等待服务端回复，这样效率会较低，网络流量吞吐率达不到高速传输效果。

2）如果不采用同步机制，无法保证数据传输结果是否正确，这时如何进行数据重发，重发次数如何保证。

3）数据量大时，服务不需要关心分包，协议层如何进行分包处理。

4）UDP 每次传输路由可能不一致，需要解决到达包的排序问题。

5.4.1 UDP 分包排序

在报文头中新增序号字段，每包数据按照序号进行排序，收到包后，根据序号进行排序。对于较大数据包，又必须标记到同一个数据区时，在报文头中采用分包字段进行分包，分包采用两个字节，第一个字节记录分包个数，第二个字节标记分包序号。

5.4.2 UDP 心跳检测

UDP 客户端定时(配置项 retry_interval)向服务端发送心跳包，发送心跳包如果超过配置文件中设置的最大次数(max_not_send_period)，则需要向告警模块上报 UDP 网络超时告警。如果是服务端，在最大次数(max_not_send_period)未收到客户端的心跳包，也需要向告警模块上报 UDP 网络超时告警。

5.4.3 UDP 发包缓存

为了保证 UDP 能够重发，需要对 UDP 已发包进行暂时缓存，这样在发包失败的情况下，需要从缓存中获取数据。缓存时间从配置文件(data_save_time)中获取，超过缓存时间的数据，需要清理掉。

对于多包数据，如果数据包较小，也需要做暂缓处理，避免频繁发送数据，对于这种内部流程处理，且时间极短，不做配置处理，内部定位暂缓时间 2ms。

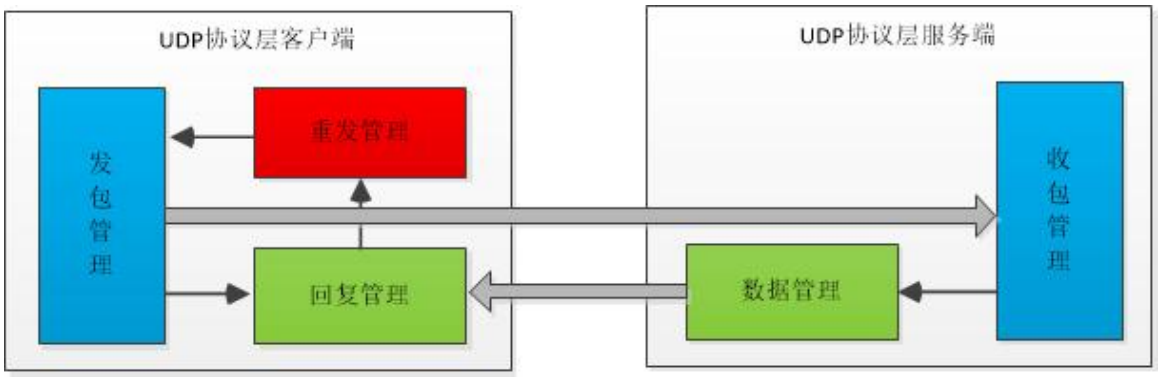
5.4.4 UDP 重发机制

UDP 重发机制采用三个模块组件来管理，发包管理模块负责处理发包数据，将发包数据按照数量大小进行分包处理，发包后将对应的套接字交给回复管理处理后，扫描重发管理组件是否又需要重发的数据，没有则继续发包，而回复管理则等待套接字对端回复，通过配置文件设置超时时间，如果在超时时间内，收到回复，并且是成功标识，则不丢给重发管理组

件，如果回复管理组件套接字超时或回复数据是错误，则将该包序号保存到重发管理组件，重发管理组件检查该包重发次数，如果超过重发次数，则上报告警并丢掉给包，否则保存到重发队列中。

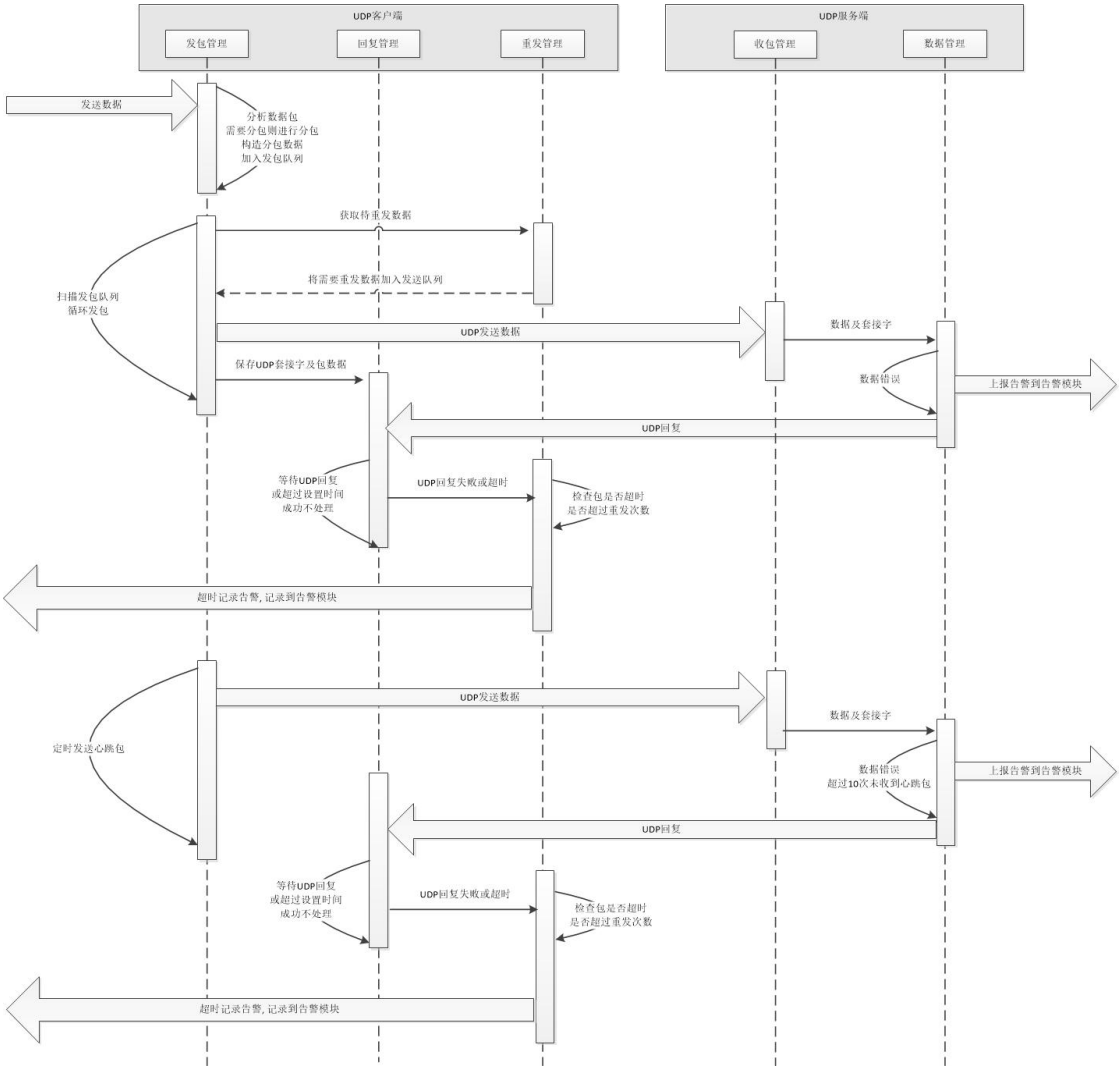
5.4.5 UDP 协议层框架图：

UDP 协议分为客户端和服务端，客户端专注发送数据，服务端专注接收数据，客户端需要服务端回复，但不需要同步处理，即发包管理值负责批量发送数据，回复管理由单独线程处理，服务端收包管理只负责接收数据，数据校验和处理以及回复由单独的数据管理线程去处理，这样可以有效提高数据发送效率。



5.4.6 UDP 协议层流程图：

整个流程按照模块组件设计，单独封装到协议层组件中，作为动态链接库，提供给边缘网关和服务使用，这样边缘网关和服务可以不用关心数据处理的流程，专注于业务和数据的处理。



UDP 配置文件：

```
#网络配置，客户端数量，网络字节序，1 表示网络字节序，2 表示大字节序，3 表示小字节序
[Network]
client_num=2
net_order=1

#远程客户端地址，IP 和 PORT 与序号对应
[RemoteAddress]
ip_1=192.168.0.1
port_1=9536
ip_2=192.168.0.2
port_2=9536

#本地 IP
[LocalAddress]
ip=192.168.0.3

#配置模式，重试次数根据网络情况而定，最大发送间隔为状态检测间隔时间。超时时间单位为秒
[Mode]
retry_type=0
retry_interval=10
max_not_send_period=10
retry_count=3
time_out=60
data_save_time=120
```

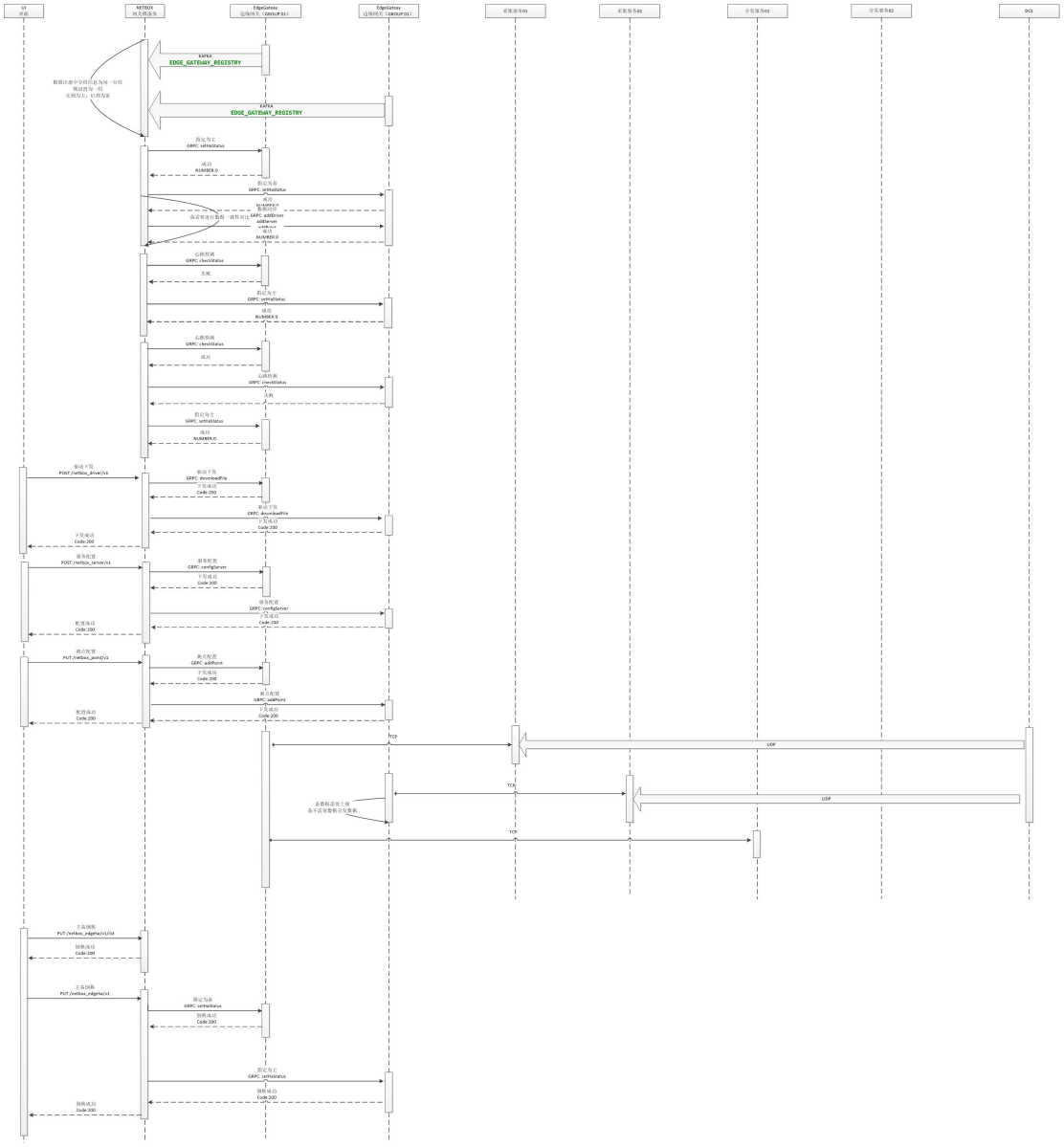
6. 主备冗余

边缘网关冗余采用中心侧网关管理微服务来管控，主要有两种方式：

一、通过心跳检测机制自动控制主备倒换，初次确定主网关，采用先后上报边缘网关注册的作为主。

二、通过界面采用倒换菜单进行手动主备倒换。

6.1 整体流程设计；



6.2 中心侧主备冗余方案设计：

边缘网关上报时需要填写分组信息，在上报注册的 KAFKA 信息中新增分组字段。中心侧接收到边缘网关上报信息后，根据分组检查是否存在同组的其他边缘网关，如果不存在，则当前边缘网关设置为主，并通过 GRPC 下发主的配置命令。如果分组中已经存在主边缘网关，并且主边缘网关的状态为正常状态，则将当前边缘网关设置为备，也需要通过 GRPC 下发备

的配置命令给边缘网关。如果原有的边缘网关状态不正常（离线或 GRPC 心跳检测失败），则需要将当前边缘网关作为主，并通过 GRPC 下发主的配置命令。

当边缘网关上报信息中没有分组信息，则该边缘网关不支持主备倒换，不需要下发设置主备的 GRPC 信息。

中心侧在边缘网关管理界面增加一个主备状态列，点击该列通过查询后端主备列表，获取当前边缘网关的主备信息。在中心侧管理界面边缘网关如果存在主备，则显示边缘网关名称为分组名称，如果不存在主备状态，则显示边缘网关名称。在操作中增加一项为手动切换主备，当点击该菜单时，调用手动切换主备接口进行主备倒换。

中心侧下发主备倒换时，会将主的最后一次心跳检测时间带上，备需要根据该时间进行相应的数据转发。

在主或备上线时，需要进行一次数据同步，这时的数据同步以中心侧为主，需要进行驱动对比，服务对比和测点对比，如果不一致，则需要将中心侧数据下发给上线的边缘网关。即如果边缘网关启动时，中心侧发现没有分组内的其他边缘网关，则需要按边缘网关数据为准，同步中心侧数据。如果中心侧发现已经存在主边缘网关，则以中心侧数据为准，同步给边缘网关。

1) 接口设计

接口包括针对界面能够查询主备列表信息，手动切换主备，边缘侧上报给中心侧注册信息中需要新增分组信息，中心侧对边缘侧进行切换的 GRPC 指令。

A) 中心侧 RESTFUL 接口：

```
/netbox_edgeha/v1:
  put:
    tags:
      - "netboxEdgeha"
    summary: "主备切换"
    operationId: "switchHost"
    produces:
      - "*/*"
      - "application/json"
    consumes:
      - "*/*"
      - "application/json"
    parameters:
      - name: "edgeHaStatus"
        in: body
        description: "主备冗余信息"
        required: true
        schema:
          $ref: '#/definitions/EdgeHaStatus'
    responses:
      200:
        description: "手动切换主备"
        schema:
          $ref: '#/definitions/RetInfo'
      400:
        description: "切换失败"
        schema:
          $ref: '#/definitions/RetInfo'
      404:
        description: "请求不存在"
  get:
    tags:
      - "netboxEdgeha"
    summary: "导出测点"
    operationId: "getEdgeha"
    produces:
      - "*/*"
      - "application/json"
    consumes:
      - "*/*"
      - "application/json"
    parameters:
      - name: "edgeHaStatus"
        in: body
        description: "主备冗余信息"
        required: true
        schema:
          $ref: '#/definitions/EdgeHaStatus'
    responses:
      200:
        description: "主备列表"
        schema:
          $ref: '#/definitions/EdgeHaStatusRsp'
      400:
        description: "失败"
        schema:
          $ref: '#/definitions/RetInfo'
      404:
        description: "请求不存在"
```

```
/netbox_edgeha/v1/list:
  get:
    tags:
      - "netboxEdgeha"
    summary: "查看网关主备冗余列表"
    operationId: "listHa"
    produces:
      - "application/json"
    consumes:
      - "*/*"
      - "application/json"
    parameters:
      - name: "edgeHaStatus"
        in: body
        description: "主备信息"
        required: true
        schema:
          $ref: '#/definitions/EdgeHaStatus'
    responses:
      200:
        description: "主备冗余列表"
        schema:
          $ref: '#/definitions/EdgeHaStatusRsp'
      400:
        description: "删除失败"
        schema:
          $ref: '#/definitions/RetInfo'
      404:
        description: "请求不存在"
```

```
EdgeHaStatus:
  allOf:
    - $ref: "#/definitions/BaseEntity"
    - type: "object"
      properties:
        id:
          description: "主备状态 ID"
          type: "integer"
          format: "int64"
        netboxId:
          description: "中心侧网关 ID"
          type: "integer"
          format: "int64"
        edgeId:
          description: "边缘网关 ID"
          type: "integer"
          format: "int64"
        groupId:
          description: "分组 ID"
          type: "integer"
          format: "int64"
        groupName:
          description: "分组名称"
          type: "string"
        groupCode:
          description: "分组编码"
          type: "string"
        groupType:
          description: "分组类型: 0-主备,
1-全主, 2-全备"
          type: "integer"
          format: "int32"
        memberLimit:
          description: "成员数量"
          type: "integer"
          format: "int32"
        hostType:
          description: "主备类型: 0-备, 1-
主"
          type: "integer"
          format: "int32"
        remark:
          description: "备注"
          type: "string"
        additional:
          description: "附加信息"
          type: "string"

EdgeHaStatusRsp:
  allOf:
    - $ref: "#/definitions/RetInfo"
    - type: "object"
      properties:
        rows:
          type: array
          items:
            $ref:
'#/definitions/EdgeHaStatus'
        total:
          type: "integer"
          format: "int64"
```

B) 边缘侧 KAFKA 接口：

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
import "edgehagroup.proto";
package edge;

message EdgeGatewayInfo {
  required string ServiceName=1; //边缘网关名称，用于在主动上报时，区分边缘网关
  required string ServiceCode=2; //网关编码
  required int32  Type=3;        //网关类型
  required string Ip=4;          //IP 地址
  required int32  Port=5;        //GRPC 连接端口
  required int32  FilePort=6;    //文件传输连接端口
  required int32  Status=7;      //边缘网关状态
  required string Version=8;      //边缘网关版本
  required string Remark=9;      //备注
  optional string Additional=10;    //附加信息
  optional EdgeHaGroupInfo HaInfo=11; //主备分组信息
}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
 //option java_outer_classname = "";
package edge;

message EdgeHaGroupInfo {
  required string GroupName=1;   //边缘网关分组名称
  required string GroupCode=2;   //分组编码
  required int32  GroupType=3;   //分组类型 0-主备，1-全主,2-全备
  required string MemberLimit=4; //分组成员限制
  required int32  HaStatus=5;    //默认主备状态 0-备，1-主
  required string Remark=6;      //备注
  optional string Additional=7;    //附加信息
}
```

C) 边缘侧 GRPC 接口：

//设置主备状态

```
rpc setHaStatus(HaInfo) returns(ErrCode) {}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message HaInfo {
  required string EdgeName=1;        //网关名称
  required string EdgeCode=2;         //网关编码
  required string GroupName=3;        //分组名称
  required string GroupCode=4;        //分组编码
  required int32  HaStatus=5;        //主备状态，0-备，1-主
  required int32  SwitchTime=6;      //切换任务开始时间
  required string Remark=7;          //备注
  optional string Additional=8;       //附加信息
}
```

2) 数据库设计

d) 主备分组表(netbox_hagroup)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	group_name	bigint(20)	分组名称
3	group_type	int(4)	分组类型：0-主备，1-全主，2-全备
4	member_limit	int(4)	成员个数，0 为无限制
5	group_code	varchar(6)	分组编码，相同名称的分组验证码必须一致
6	del_flag	int(4)	删除标记：0-未删除，1-删除
7	create_by	varchar(60)	创建者
8	create_time	datetime	创建时间
9	update_by	varchar(60)	更新者
10	update_time	datetime	更新时间
11	remark	varchar(1000)	备注信息
12	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

e) 主备关系表(netbox_harelation)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值

2	edge_id	bigint(20)	边缘网关 ID
3	group_id	bigint(20)	分组 ID
4	host_type	int(4)	主备类型：0-备，1-主
5	del_flag	int(4)	删除标记：0-未删除，1-删除
6	create_by	varchar(60)	创建者
7	create_time	datetime	创建时间
8	update_by	varchar(60)	更新者
9	update_time	datetime	更新时间
10	remark	varchar(1000)	备注信息
11	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

6.3 边缘网关主备冗余方案设计：

边缘网关启动时，需要根据配置文件上报注册信息，注册信息中包括分组信息。启动时默认为备，只有接收到网关设定的 GRPC 指定为主才切换为主状态。

边缘网关为主时，需要接收采集器的数据，将数据持久化到 IOTDB 中，并根据分发器服务订阅将对应的测点信息转发给对应的分发器。

边缘网关为备时，同样需要接收采集器的数据，也需要将数据持久化到 IOTDB 中，但是不需要将数据转发给分发器。

边缘网关启动时，需要根据配置文件获取分组信息，如果没有分组，则不上报分组给中心侧，中心侧获取不到分组，表明该边缘网关不支持主备，不下发主备信息。边缘侧对于不配置分组，则当默认为主处理。如果有分组信息，并且分组信息中默认状态为备，则上报备状态，并设置当前状态为备，等待中心侧设置主备信息。如果为主，则将当前设置为主，当接收到边缘侧为备时，才切换到备状态。

1) 接口设计

边缘网关的主备只保留在网关上，不下发到服务。

//设置主备状态

```
rpc setHaStatus(HaInfo) returns(ErrCode) {}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message HaInfo {
  required string EdgeName=1;         //网关名称
  required string EdgeCode=2;          //网关编码
  required string GroupName=3;         //分组名称
  required string GroupCode=4;         //分组编码
  required int32  HaStatus=5;         //主备状态，0-备，1-主
  required int32  SwitchTime=6;       //切换任务开始时间
  required string Remark=7;           //备注
  optional string Additional=8;        //附加信息
}
```

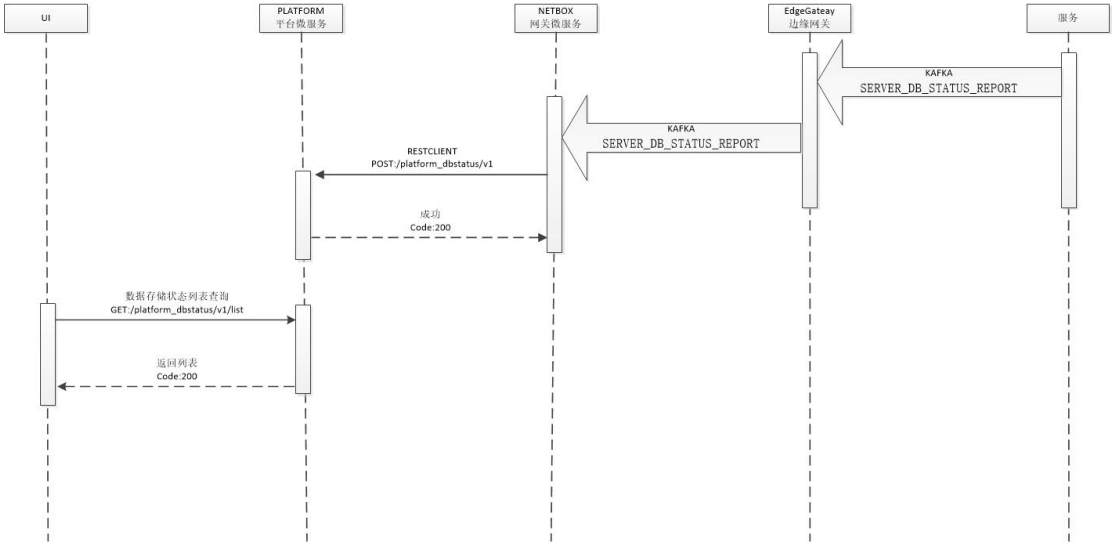
2) 配置设计

在边缘网关配置信息文件 config.ini 中，增加分组配置项，如果没有分组，则不需要配置主备，不上报分组信息。

```
GW_GROUP_NAME=QW-01

GW_GROUP_CODE=GROUP-00000-0000-0000-00001

GW_GROUP_TYPE=0

GW_GROUP_MEMBER_LIMIT=2

GW_GROUP_HA_STATUS=0
```

7. 性能统计及流量检测

7.1 数据存储状态上报



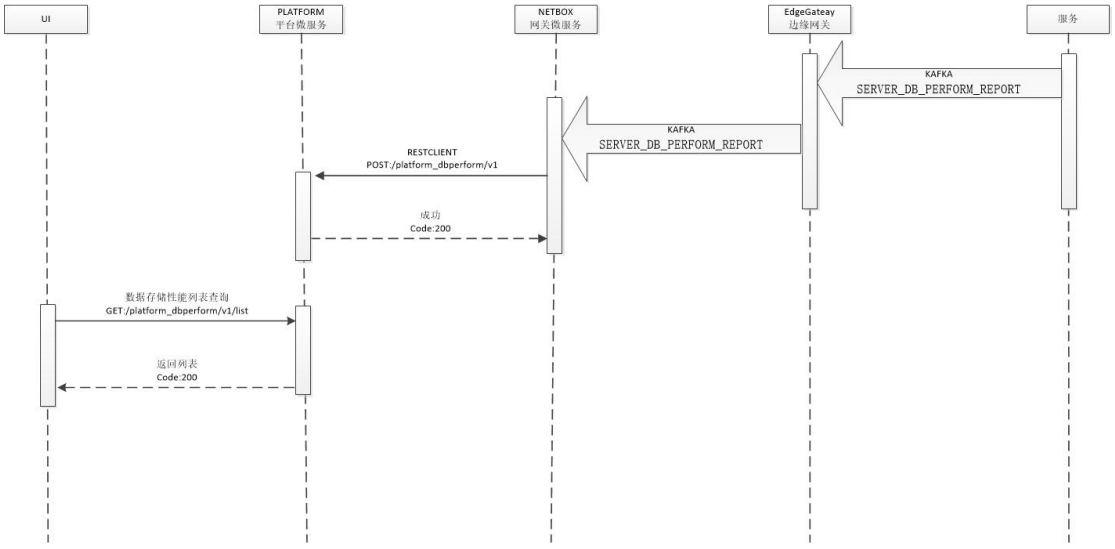
服务收集数据库状态，通过 KAFKA 上报给中心侧网关。上报内容包括服务信息，数据库信息，数据库操作类型，操作结果，附件信息填写操作失败的具体错误信息。

TOPIC: SERVER_DB_STATUS_REPORT

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message DbStatistics {
  required int32  DbOperate=1;         //0-插入 1-查询 2-更新 3-删除 4-连接 5-未知
  required int64  DataCount=2;         //数量
  required int32  Result=3;            //结果，0-成功，1-失败
  required int64  UseTime=4;           //操作执行时间
  required int64  TimeRange=5;         //统计时间范围
  required string Remark=6;            //备注
  optional string Additional=7;        //附加信息
}
```

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
mport "dbstatistics.proto";
package edge;

message DbStatusInfo {
  required string ServerName=1;         //服务名称
  required string ServerCode=2;         //服务编码
  required int32  ServerType=3;         //服务类型 0-采集 1-分发
  required int32  DbType=4;             //数据库类型 0-未知 1-IOTDB 2-PI 3-SQLITE 4-MYSQL
  repeated DbStatistics Statistics=5;   //数据库统计信息
  required string DbIp=6;               //数据库 IP 地址
  required int32  DbPort=7;             //数据库端口
  required string DbParam=8;            //连接参数
  required int64 ReportTime=9;          //上报时间
  required string Remark=10;            //备注
  optional string Additional=11;        //附加信息，填写错误信息
}
```

7.2 数据库性能上报

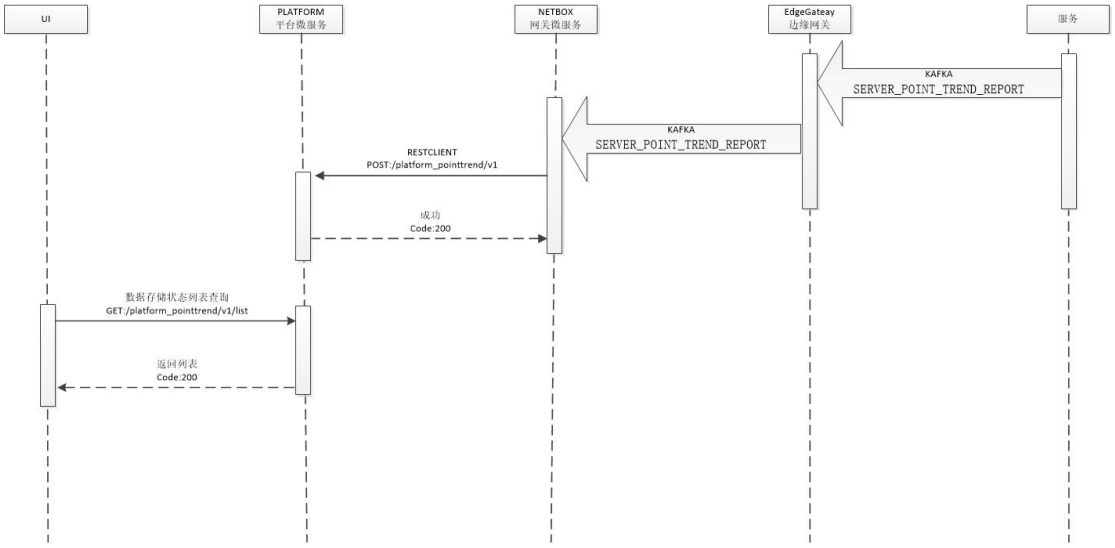


上报数据处理性能数据，包括服务信息，数据库信息，操作类型，操作数据条数，耗时等。

TOPIC: SERVER_DB_PERFORM_REPORT

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
import "dbcount.proto";
package edge;

message DbPerformInfo {
  required string ServerName=1;        //服务名称
  required string ServerCode=2;        //服务编码
  required int32  ServerType=3;        //服务类型 0-采集 1-分发
  required int32  DbType=4;            //数据库类型 0-未知 1-IOTDB 2-PI 3-SQLITE 4-MYSQL
  repeated DbStatistics Statistics=5;  //数据库统计信息
  required int64 ReportTime=6;         //上报时间
  required string Remark=7;            //备注
  optional string Additional=8;        //附加信息
}
```

7.3 测点趋势上报



需要上报在时间范围内服务信息，测点信息，测点接收数据数量。

TOPIC: SERVER_POINT_TREND_REPORT

```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
package edge;

message PointTrend {
  required string PointName=1;        //测点名称
  required string PointCode=2;        //测点编码
  required int32  PointType=3;        //测点类型 0-boolean 1-int32 2-int64 3-float 4-double 5-text
  required int64  ReportTime=4;       //统计截止时间
  required int64  TimeRange=5;        //统计时间范围
  required int64  RecordCount=6;      //接收数据条数
  required string Remark=7;           //备注
  optional string Additional=8;       //附加信息
}
```

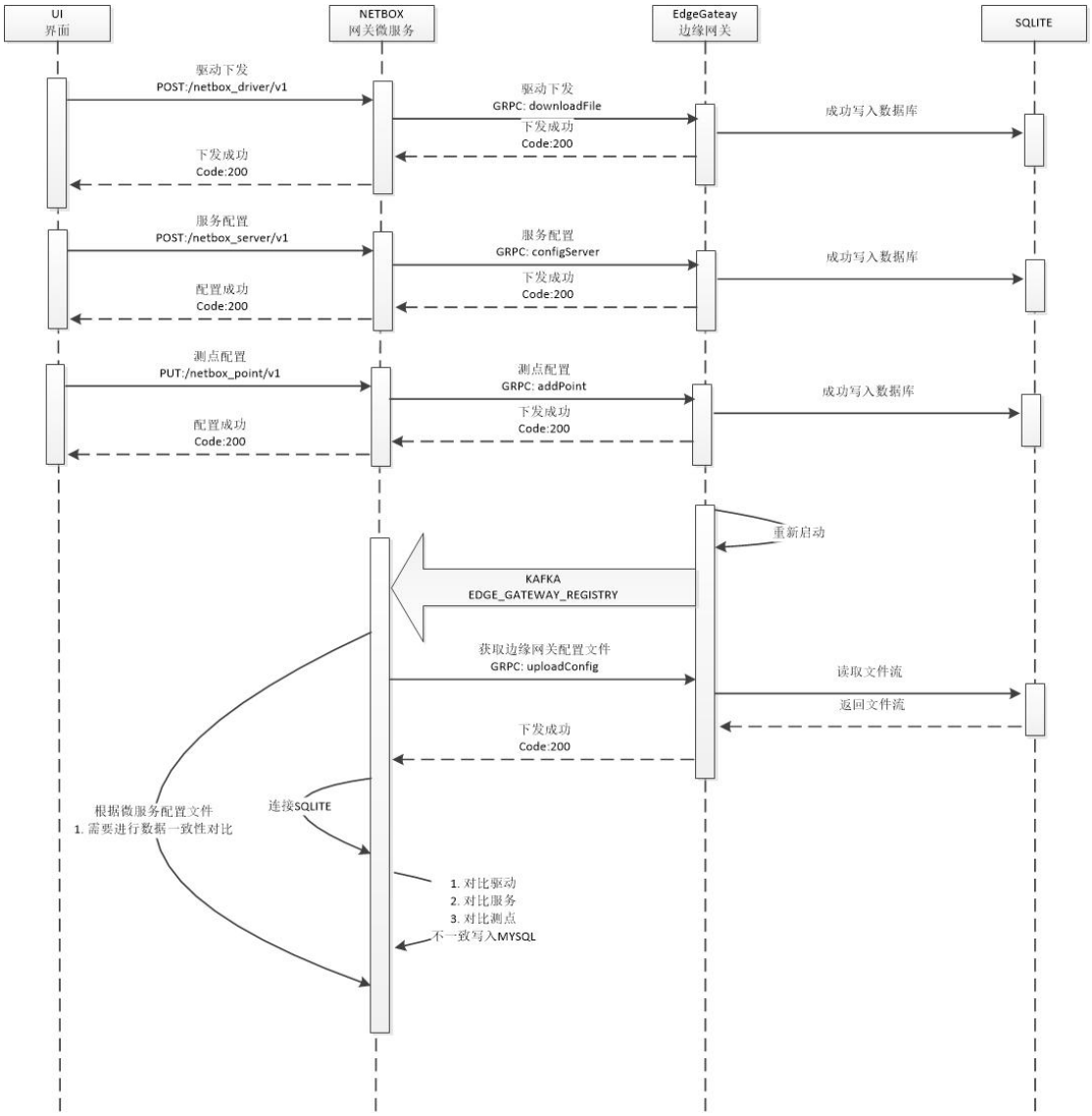
```
syntax = "proto2";
option java_package = "com.cnnp.netbox.kafka.msg";
//option java_outer_classname = "";
import "pointtrend.proto";
package edge;

message PointTrendInfo {
  required string ServerName=1;        //服务名称
  required string ServerCode=2;        //服务编码
  required int32  ServerType=3;        //服务类型 0-采集 1-分发
  repeated PointTrend Points=4;        //测点趋势
  required string ReportTime=5;         //上报时间
  required string Remark=6;           //备注
  optional string Additional=7;        //附加信息
}
```

PROTO 文件：

dbperforminfo.proto　dbstatistics.proto　dbstatusinfo.proto　pointtrend.proto

8. 边缘网关数据一致性

8.1 整体流程设计；



8.2 边缘网关数据持久化：

1）边缘网关在接收到中心网关的配置消息时，需要保存配置信息，具体包括驱动下载，服务配置，测点配置等。

2）数据持久化外置数据库采用 sqlite 文件型关系数据库。

3）数据库设计

f) 驱动表(netbox_driver)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	driver_name	varchar(200)	驱动名称
3	driver_type	int(4)	驱动类型
5	version	varchar(30)	驱动版本
6	file_name	varchar(200)	文件名称
7	file_type	int(4)	文件类型
8	file_path	varchar(300)	文件保存路径
9	status	int(4)	使用状态：0-未使用，1-已使用

10	create_time	datetime	创建时间
11	remark	varchar(1000)	备注信息
12	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

g) 服务表(netbox_server)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	server_name	varchar(200)	服务名称
3	server_code	varchar(200)	服务编码
4	server_type	int(4)	服务类型
5	status	int(4)	使用状态：0-未启动，1-已启动，2-停止
6	create_time	datetime	创建时间
7	driver_id	bigint(20)	驱动 ID，对应驱动表中的 id
8	remark	varchar(1000)	备注信息
9	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

h) 测点表(netbox_point)

序号	字段名	数据类型	说明
1	id	bigint(20)	主键，唯一值
2	point_name	varchar(200)	测点名称
3	point_code	varchar(200)	测点编码
4	point_type	int(4)	测点类型 0-boolean 1-int32 2-int64 3-float 4-double 5-text
5	max_value	double(8,2)	最大值
6	min_value	double(8,2)	最小值
7	attribute	varchar(60)	附加属性
8	status	int(4)	使用状态：0-未配置，1-已配置
9	create_time	datetime	创建时间
10	server_id	bigint(20)	服务 ID，对应服务表中的 id
11	remark	varchar(1000)	备注信息
12	additional	TEXT	扩展字段，通常以 JSON 字符串传递附加信息

8.3 中心平台删除中心网关：

1）当边缘网关在线时，不需要删除边缘网关数据，只删除中心平台网关，这时边缘网关关联的其他数据不会删除，包括驱动，服务和测点；

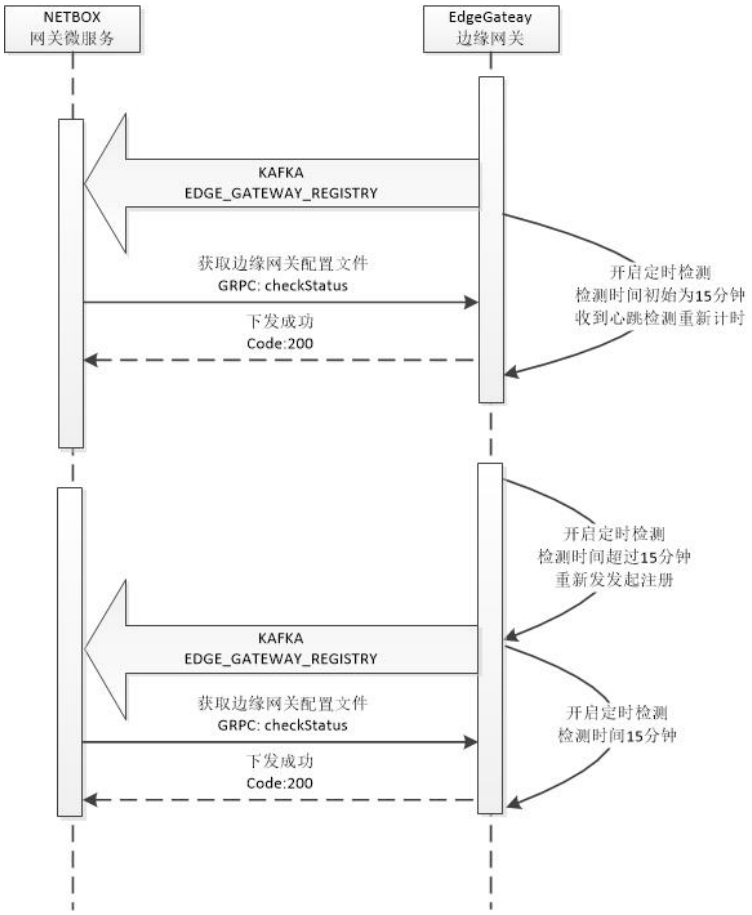
2）当边缘网关离线时，会自动删除边缘网关，这时也需要删除边缘网关所在的驱动，服务和测点；同时也需要更新驱动状态和测点状态；

1. 边缘网关注册时自动对比：

1）边缘网关注册时，和之前注册流程一下，需要携带边缘网关名称（唯一标识），网关编码，GRPC 连接 IP，GRPC 连接端口。
2）中心平台收到边缘网关注册，需要检查边缘网关是否存在，如果不存在，则需要添加网关，存在则更新边缘网关状态；
3）中心侧根据微服务配置开是否需要进行数据一直性检测，中心平台需要调用 GRPC 对比接口，获取边缘网关持久化数据；
4）中心平台需要对比边缘网关数据，发现数据已经不存在，则插入数据，数据对比顺序如下：

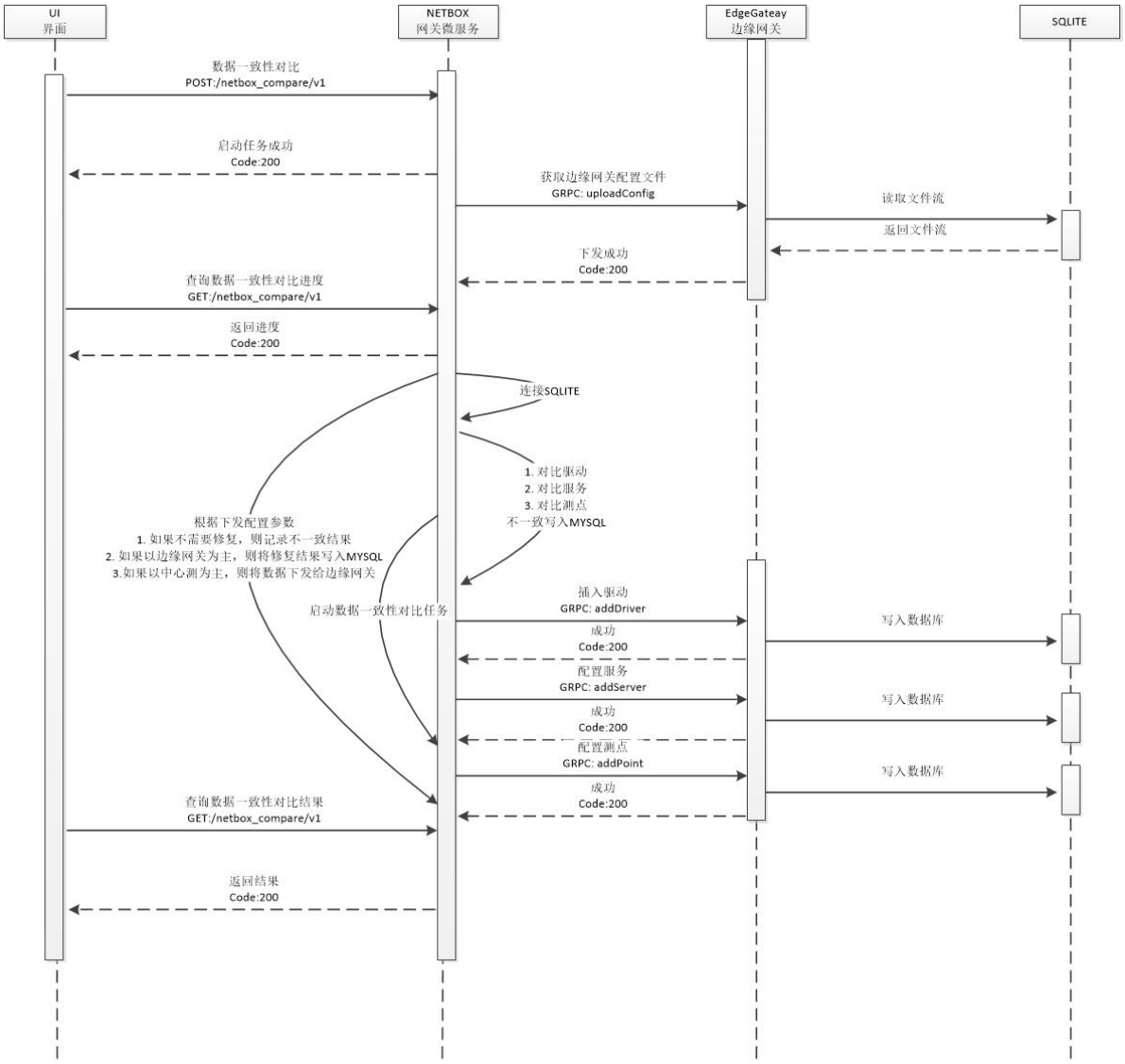
A. 对比驱动是否存在，如果不存在，则需要自动添加驱动；
B. 对比服务是否存在，不存在，需要自动添加服务；
C. 对比测点是否存在，不存在，需要自动添加测点；

5）边缘网关接收心跳检测，如果超过 15 分钟未收到心跳检测，需要再次向中心平台发起注册。



8.4 中心平台手动发起对比（离线配置，在线上载下载）：
1）在边缘网关管理操作中新增操作“数据对比”；
2）增加选项；0-只上报结果，1-以中心平台为准，2-以边缘网关为准
3）如果不需要配置，则只进行数据对比，对比结果缓存，等待前端查询，前端需要定时查询后端对比进度，如果对比结束，则显示对比结果；
4）对比接口如下：

5）流程如下



8.5 中心侧 RESTFUL 接口：

中心侧网关服务提供两个接口给界面，一个是启动对比的接口，一个是查询对比进度及结果的接口。

```yaml
/netbox_compare/v1:
  post:
    tags:
      - "netboxCompare"
    summary: "数据一致性对比"
    operationId: "startCompare"
    produces:
      - "*/*"
      - "application/json"
    consumes:
      - "*/*"
      - "application/json"
    parameters:
      - name: "netboxCompare"
        in: body
        description: "网关服务信息"
        required: true
        schema:
          $ref: '#/definitions/NetboxCompare'
    responses:
      200:
        description: "导出结果"
        schema:
          $ref: '#/definitions/RetInfo'
      400:
        description: "查询失败"
        schema:
          $ref: '#/definitions/RetInfo'
      404:
        description: "请求不存在"
  get:
    tags:
      - "netboxCompare"
    summary: "导出测点"
    operationId: "listCompare"
    produces:
      - "*/*"
      - "application/json"
    consumes:
      - "*/*"
      - "application/json"
    parameters:
      - name: "netboxCompare"
        in: body
        description: "网关服务信息"
        required: true
        schema:
          $ref: '#/definitions/NetboxCompare'
    responses:
      200:
        description: "对比结果列表"
        schema:
          $ref: '#/definitions/NetboxCompareRsp'
      400:
        description: "失败"
        schema:
          $ref: '#/definitions/RetInfo'
      404:
        description: "请求不存在"
```

```yaml
NetboxCompare:
  allOf:
    - $ref: "#/definitions/BaseEntity"
    - type: "object"
      properties:
        compId:
          description: "网关管理中下发的测点 ID，键值"
          type: "integer"
          format: "int64"
        compType:
          description: "数据类型，"
          type: "string"
        isSame:
          description: "是否一致，0-一致，1-部分一致，2-中心测有，3-边缘测有"
          type: "integer"
          format: "int32"
        diffValue:
          description: "不一致值"
          type: "string"
        recoverType:
          description: "回复方式，0-不恢复，1-以中心测为准，2-以边缘侧为准"
          type: "integer"
          format: "int64"
        recoverResult:
          description: "回复结果，0-未恢复，1-成功，2-失败"
          type: "integer"
          format: "int64"
        remark:
          description: "备注"
          type: "string"
        additional:
          description: "附加信息"
          type: "string"

NetboxCompareRsp:
  allOf:
    - $ref: "#/definitions/RetInfo"
    - type: "object"
      properties:
        rows:
          type: array
          items:
            $ref: '#/definitions/NetboxCompare'
        total:
          type: "integer"
          format: "int64"
        progress:
          type: "integer"
          format: "int64"
```

8.6 边缘网关上传数据给中心侧网关微服务的 GRPC 接口：

```
//数据库上传
rpc uploadConfig(ConfigInfo) returns(stream ConfigFile) {}


syntax = "proto2";
option java_package = "com.cnnp.netbox.grpc.msg";
package edge;

message ConfigInfo {
  required int32  Type=1;               //0-所有，1-驱动，2-服务，3-测点
  optional string FileType=2;           //文件类型，0-db,1-zip,2-csv
  optional string StartTime=3;          //开始时间
  optional string EndTime=4;            //结束时间
  optional string Remark=5;             //备注
  optional string Additional=6;         //附加信息
}

message ConfigFile {
  required bytes buffer=1;      //文件流
}
```

8.7 边缘网关创建数据库表定义的 SQL 语句：

```
DROP TABLE IF EXISTS netbox_driver;
CREATE TABLE netbox_driver (
    id INTEGER PRIMARY KEY      AUTOINCREMENT,
    driver_name             VARCHAR(200)    NOT NULL,
    driver_type             INTEGER    NOT NULL,
    version                     VARCHAR(30) ,
    file_name                VARCHAR(200) ,
    file_type                  INTEGER    NOT NULL,
    file_path                  VARCHAR(300)            NOT NULL,
    status                         INTEGER    NOT NULL,
    create_time             DATETIME,
    remark                      VARCHAR(300) ,
    additional              TEXT
) ;

DROP TABLE IF EXISTS netbox_server;
CREATE TABLE netbox_server (
    id INTEGER PRIMARY KEY      AUTOINCREMENT,
    server_name             VARCHAR(200)    NOT NULL,
    server_code             VARCHAR(200)    NOT NULL,
    server_type             INTEGER    NOT NULL ,
    driver_id                   INTEGER    NOT NULL,
    status                          INTEGER    NOT NULL,
    create_time             DATETIME,
    remark                      VARCHAR(300) ,
    additional              TEXT
) ;

DROP TABLE IF EXISTS netbox_point;
CREATE TABLE netbox_point (
    id INTEGER PRIMARY KEY      AUTOINCREMENT,
    point_name              VARCHAR(200)    NOT NULL,
    point_code              VARCHAR(200)    NOT NULL,
    point_type              INTEGER    NOT NULL ,
    max_value               REAL,
    min_value                REAL,
    attribute                  VARCHAR(60) ,
    server_id                  INTEGER    NOT NULL,
    status                          INTEGER    NOT NULL,
    create_time             DATETIME,
    remark                      VARCHAR(300) ,
    additional              TEXT
) ;
```