Rockchip Linux应用开发基础

文件标识：RK-FB-YF-358

发布版本：V1.1.0

日期：2020-06-04

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文档提供Linux应用开发基础说明。

产品版本

芯片名称	内核版本
RV1109	Linux 4.19
RK1806	Linux 4.19

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师
软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	Fenrir Lin	2020-04-28	初始版本
V1.1.0	Fenrir Lin	2020-06-04	增加ispserver和onvif_server部分

目录

1 简介：

1.1 应用

主要应用路径位于SDK工程的app路径下，对应功能如下：

应用名称	模块功能
ipcweb-ng	web前端工程
ipcweb-backend	web后端工程
ipc-daemon	系统管理及守护以下应用
storage_manager	存储管理
netserver	网络服务
mediaserver	多媒体服务
dbserver	数据库服务
ispserver	图像信号处理服务端
onvif_server	onvif协议服务端

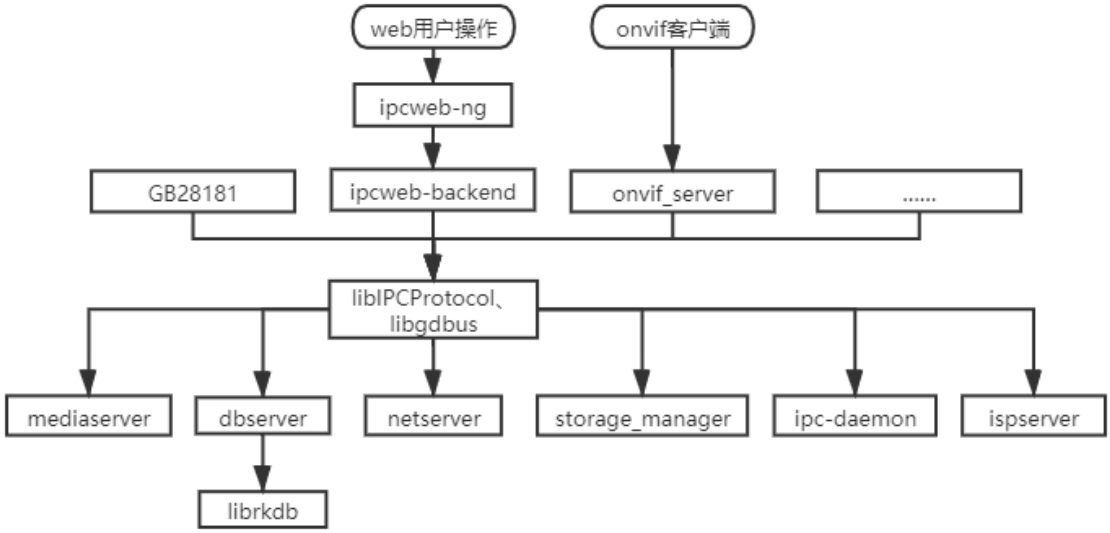
1.2 库

主要库路径位于SDK工程的app路径下，采用dbus的进程间通信机制。主要开发libIPCProtocol即可。

库名称	主要功能
libIPCProtocol	基于dbus，提供进程间通信接口，以便跨进程调用函数。
librkdb	基于sql，提供对数据库操作的接口。
libgdbus	提供dbus支持。

1.3 应用框架

应用框架如下：



目前支持以下两种方式：

1. web前端根据用户的操作，使用GET/PUT/POST/DELETE四种方法，调用不同的web后端接口。web后端中，使用libIPCProtocol提供的函数，通过dbus进行跨进程间通信，来调用相应的服务。
2. onvif客户端或支持onvif协议的NVR，通过onvif标准协议，可以直接调用onvif_server的接口，使用libIPCProtocol提供的函数，来调用相应的服务。

具体服务中，会根据传入的参数，进行相应的操作，从而使用户的操作生效。

除了以上两种方式外，还可新增GB28181协议等其他方式。此框架可以兼容不同应用，可以不耦合。

2 数据流

数据流主要为web前后端之间的http协议和dbus总线上的通信数据。统一使用JSON格式。

2.1 GET

以获取当前网卡信息为例，首先web端在进入配置-网络-基础设置时，会自动刷新，并向web后端发送一条请求，摘要如下：

```
Request URL: http://{{板端IP地址}}/cgi-bin/entry.cgi/network/lan
Request Method: GET
```

web后端收到这条Request后，会判断URL和Method的信息，调用libIPCProtocol提供的netserver_get_networkip函数，向dbus总线发送一条如下的消息**(可使用dbus-monitor工具监控)**：

```
method call time=1588045737.411643 sender=:1.371 ->
destination=rockchip.netserver serial=2 path=/;
interface=rockchip.netserver.server; member=GetNetworkIP
   string "eth0"
```

netserver服务收到这条消息后，去获取当前eth0网卡的IP地址，再通过dbus进行回复。

```
method return time=1588045737.419339 sender=:1.4 -> destination=:1.371
serial=357 reply_serial=2
   string "[ { "link": { "sInterface": "eth0", "sAddress": "72:94:20:67:b4:b8",
"iNicSpeed": 1000, "iDuplex": 1, "sDNS1": "10.10.10.188", "sDNS2": "58.22.96.66"
}, "ipv4": { "sV4Address": "172.16.21.204", "sV4Netmask": "255.255.255.0",
"sV4Gateway": "172.16.21.1" }, "dbconfig": { "sV4Method": "dhcp", "sV4Address":
"", "sV4Netmask": "", "sV4Gateway": "", "sDNS1": "", "sDNS2": "" } } ]"
```

web后端对dbus消息进行处理后，通过http发送以下信息给web前端，进而显示在网页界面上。

```
response:
{
    "ipv4": {
        "sV4Address": "172.16.21.204",
        "sV4Gateway": "172.16.21.1",
        "sV4Method": "dhcp",
        "sV4Netmask": "255.255.255.0"
    },
    "link": {
        "sAddress": "72:94:20:67:b4:b8",
        "sDNS1": "10.10.10.188",
        "sDNS2": "58.22.96.66",
        "sInterface": "eth0",
        "sNicType": "1000MD"
    }
}
```

2.2 PUT

以设置IP地址为例，web前端发送如下请求：

```
Request URL: http://172.16.21.204/cgi-bin/entry.cgi/network/lan
Request Method: PUT
Request Payload:
{
    "ipv4": {
        "sV4Address": "172.16.21.205",
        "sV4Gateway": "172.16.21.1",
        "sV4Method": "manual",
        "sV4Netmask": "255.255.255.0"
    },
    "link": {
        "sAddress": "72:94:20:67:b4:b8",
        "sInterface": "eth0",
        "sNicType": "1000MD",
        "sDNS1": "10.10.10.188",
        "sDNS2": "58.22.96.66"
    }
}
```

web后端调用libIPCProtocol提供的dbserver_network_ipv4_set函数，向dbus总线发送如下消息

```
method call time=1588054078.249193 sender=:1.447 ->
destination=rockchip.dbserver serial=3 path=/; interface=rockchip.dbserver.net;
member=Cmd
   string "{ "table": "NetworkIP", "key": { "sInterface": "eth0" }, "data": {
"sV4Method": "manual", "sV4Address": "172.16.21.205", "sV4Netmask":
"255.255.255.0", "sV4Gateway": "172.16.21.1" }, "cmd": "Update" }"
```

该消息发往的interface是**rockchip.dbserver.net**，将会update数据库中NetworkIP这张表的数据。同时netserver会监听到此interface的广播，根据消息中的IP地址去进行设置。

web后端的部分接口会再获取一次最新值，返还给前端。

3 ipcweb-ng

3.1 开发基础

web前端，采用Angular 8框架。

开发语言： Typescript，JavaScript，HTML5，CSS3

参考文档:

Angular官方入门教程
TypeScript中文网
w3school

代码路径： app/ipcweb-ng

编译命令：

```
#在app/ipcweb-ng目录下
ng build --prod
#将编译生成在app/ipcweb-ng/dist目录下的文件，都移动到device/rockchip/oem/oem_ipc/www路径下
#在SDK根目录下
make rk_oem-dirclean && make rk_oem #重新编译oem
./mkfirmware.sh #打包oem.img,再进行烧写
```

3.2 开发环境

```
sudo apt update
sudo apt install nodejs
sudo apt install npm
sudo npm install -g n # 安装 n 模块
sudo n stable # 用 n 模块升级
npm npm --version # 确认 npm 版本
sudo npm install -g @angular/cli # 安装 Angular 命令行工具
```

3.3 在线调试

启动webpack开发服务

```
ng serve
```

成功的话，可见以下log

```
** Angular Live Development Server is listening on 0.0.0.0:4200, open your browser on http://localhost:4200/ **
```

随后使用chrome浏览器访问 http://localhost:4200/，即可在线调试。

也可使用`ng build --prod`命令编译，将生成在dist目录下的文件，推送到板端，替换/oem/www下的文件。如果浏览器访问页面未更新，需要清理浏览器图片和文件的缓存。

3.4 代码框架

```
src/
├── app
│ ├── about # 关于页面，项目说明文字
│ ├── app.component.html # 应用主入口
│ ├── app.component.scss # scss 样式文件
│ ├── app.component.spec.ts # 测试 spec 文件
│ ├── app.component.ts # app 组件
│ ├── app.module.ts # app 模块
│ ├── app-routing.module.ts # 主路由
│ ├── auth # 认证模块，包括登录页面，用户认证
│ ├── config # 配置模块，包含所有配置子组件
│ ├── config.service.spec.ts # 配置模块测试 spec 文件
│ ├── config.service.ts # 配置模块服务，用于与设备通信以及模块间通信
│ ├── footer # footer 模块，版权声明
│ ├── header # header 模块，导航路由，用户登录/登出
│ └── preview # 预览模块，主页面码流播放器
├── assets
│ ├── css # 样式
│ ├── i18n # 多国语言翻译
│ ├── images # 图标
│ └── json # 调试用 json 数据库文件
├── environments # angular 发布环境配置
│ ├── environment.prod.ts
│ └── environment.ts
├── favicon.ico # 图标
├── index.html # 项目入口
├── main.ts # 项目入口
├── polyfills.ts
├── styles.scss # 项目总的样式配置文件
└── test.ts
14 directories, 16 files
```

详细模块位于 src/app/config

```
$ tree -L 2 src/app/config
├── config-audio # 音频配置
│ ├── config-audio.component.html
│ ├── config-audio.component.scss
│ ├── config-audio.component.spec.ts
│ └── config-audio.component.ts
├── config.component.html # config组件主页面
├── config-event # 事件配置
├── config-image # ISP图像配置
├── config-intel # Intelligent智能分析配置
├── config.module.ts # 配置模块
├── config-network # 网络配置
├── config-routing.module.ts # 配置模块子路由
├── config-storage # 存储配置
├── config-system # 系统配置
├── config-video # 视频编码配置
├── MenuGroup.ts # 菜单数据类
├── NetworkInterface.ts # 网络接口数据类
└── shared # 一些共享子模块，可复用，方便后面主模块调整
    ├── abnormal
    ├── alarm-input
    ├── alarm-output
    ├── center-tip
```

```
    ├── cloud
    ├── ddns
    ├── email
    ├── encoder-param
    ├── ftp
    ├── hard-disk-management
    ├── info
    ├── intrusion-detection
    ├── intrusion-region
    ├── isp
    ├── motion-arming
    ├── motion-detect
    ├── motion-linkage
    ├── motion-region
    ├── ntp
    ├── osd
    ├── picture-mask
    ├── port
    ├── pppoe
    ├── privacy-mask
    ├── protocol
    ├── region-crop
    ├── roi
    ├── save-tip
    ├── screenshot
    ├── smtp
    ├── tcpip
    ├── time-table
    ├── upgrade
    ├── upnp
    └── wifi
```

4 ipcweb-backend

4.1 开发基础

web后端，采用ngingx+fastcgi，调试可以使用curl、postman或者直接与web前端联调。

开发语言： C++

参考文档：

HTPP协议知识
RESTful API 规范
Nginx + CGI/FastCGI + C/Cpp
POSTMAN

代码路径： app/ipcweb-backend

编译命令：

```
#在SDK根目录下
make ipcweb-backend-dirclean && make ipcweb-backend
make rk_oem-dirclean && make rk_oem #重新编译oem
./mkfirmware.sh #打包oem.img，再进行烧写
```

配置文件：

nginx配置文件位于buildroot/board/rockchip/puma/fs-overlay/etc/nginx/nginx.conf，部分摘要如下：

```
location /cgi-bin/ {
    gzip off;
    # 网页根目录
    root /oem/www;
    fastcgi_pass unix:/run/fcgiwrap.sock;
    fastcgi_index entry.cgi;
    fastcgi_param DOCUMENT_ROOT /oem/www/cgi-bin;
    # CGI 应用唯一入口
    fastcgi_param SCRIPT_NAME /entry.cgi;
    include fastcgi_params;

    # 解决PATH_INFO变量问题
    set $path_info "";
    set $real_script_name $fastcgi_script_name;
    if ($fastcgi_script_name ~ "^(.+?\.cgi)(/.+)$") {
        set $real_script_name $1;
        set $path_info $2;
    }
    fastcgi_param PATH_INFO $path_info;
    fastcgi_param SCRIPT_FILENAME $document_root$real_script_name;
    fastcgi_param SCRIPT_NAME $real_script_name;
}
```

4.2 编译环境

可以在SDK根目录下使用make ipcweb-backend编译，也可以使用以下命令编译。

```
mkdir build && cd build
【可选】该项目使用Google Test作为测试框架。初始化googletest子模块以使用它。
  git submodule init
  git submodule update

cmake .. -DCMAKE_TOOLCHAIN_FILE=
<path_of_sdk_root>/buildroot/output/rockchip_puma/host/share/buildroot/toolchain
.cmake

make
```

4.3 调试环境

1. 将编译出的entry.cgi文件推送到设备端的/oem/www/cgi-bin/路径下，确保entry.cgi文件的权限和用户组如下：

```
-rwxr-xr-x 1 www-data www-data 235832 Apr 26 20:51 entry.cgi
```

2. 确保设备端nginx服务已经启动，可使用ps命令查看。

```
538 root     12772 S    nginx: master process /usr/sbin/nginx
539 www-data 13076 S    nginx: worker process
```

3. 使用ifconfig -a命令获取设备端的IP地址。
4. 使用curl命令进行调试，示例如下：

```
$ curl -X GET http://172.16.21.217/cgi-bin/entry.cgi/network/lan
{"ipv4":
{"sV4Address":"172.16.21.217","sV4Gateway":"172.16.21.1","sV4Method":"dhcp","sV4
Netmask":"255.255.255.0"},"link":
{"sAddress":"84:c2:e4:1b:66:d8","sDNS1":"10.10.10.188","sDNS2":"58.22.96.66","sI
nterface":"eth0","sNicType":"10MD"}}
```

5. 由于CGI不能使用标准输出流，所以log保存在以下路径。

```
$ cat /var/log/messages
# 调试log输出到syslog
$ cat /var/log/nginx/error.log
# 网页服务器错误log
$ cat /var/log/nginx/access.log
# 网页服务器访问log
```

5 ipc-daemon

5.1 开发基础

系统守护服务，提供系统维护相关服务，初始化和确保dbserver/netserver/storage_manager/mediaserver的运行。

开发语言： C

代码路径： app/ipc-daemon

编译命令： 在SDK根目录下，`make ipc-daemon-dirclean && make ipc-daemon`

5.2 对外接口

以下接口位于app/libIPCProtocol/system_manager.h中。

函数名称	函数功能
system_reboot	系统重启
system_factory_reset	恢复出厂设置
system_export_db	导出数据库
system_import_db	导入数据库
system_export_log	导出调试日志
system_upgrade	OTA固件升级

6 storage_manager

6.1 开发基础

存储管理服务，提供文件查询、硬盘管理、录像抓图配额等功能。

开发语言： C

代码路径： app/storage_manager

编译命令： 在SDK根目录下，`make storage_manager-dirclean && make storage_manager`

6.2 对外接口

以下接口位于app/libIPCProtocol/storage_manager.h中。

函数名称	函数功能
storage_manager_get_disks_status	获取硬盘状态
storage_manager_get_filelist_id	根据ID获取文件列表
storage_manager_get_filelist_path	根据路径获取文件列表
storage_manager_get_media_path	获取媒体文件路径信息
storage_manager_diskformat	硬盘格式化

7 netserver

7.1 开发基础

网络服务，提供获取网络信息，扫描Wi-Fi，配网等功能。

开发语言： C

代码路径： app/netserver

编译命令： 在SDK根目录下，`make netserver-dirclean && make netserver`

7.2 对外接口

以下接口位于app/libIPCProtocol/netserver.h中。

函数名称	函数功能
netserver_scan_wifi	扫描Wi-Fi
netserver_get_service	获取Wi-Fi或以太网的service信息
netserver_get_config	获取service对应的配置信息
netserver_get_networkip	获取eth0或wlan0的网卡信息

8 dbserver

8.1 开发基础

数据库服务，对数据库进行初始化，提供对数据库相关操作接口。

开发语言： C

代码路径： app/dbserver

编译命令： 在SDK根目录下，`make dbserver-dirclean && make dbserver`

8.2 对外接口

接口位于app/libIPCProtocol/dbserver.h中，主要对数据库不同table进行select、update、delete等操作。

8.3 调试环境

修改完代码，重新编译后，设备端需要执行以下操作：

```
killall dbserver
rm /data/sysconfig.db
#将新编译的dbserver推送进来替换
dbserver&
```

可使用以下命令，发送dbus消息来查询数据是否正常。

```
# dbus-send --system --print-reply --dest=rockchip.dbserver /
rockchip.dbserver.net.Cmd \
> string:"{ \"table\": \"ntp\", \"key\": { }, \"data\": \"*\", \"cmd\":
\"Select\" }"

method return time=1588123823.096268 sender=:1.5 -> destination=:1.6 serial=7
reply_serial=2
   string "{ "iReturn": 0, "sErrMsg": "", "jData": [ { "id": 0, "sNtpServers":
"122.224.9.29 94.130.49.186", "sTimeZone": "posix\/Etc\/GMT-8", "iAutoMode": 1,
"iRefreshTime": 60 } ] }"
```

9 mediaserver

9.1 开发基础

提供多媒体服务的主应用，具体开发请参考《MediaServer开发基础》

开发语言： C++

代码路径： app/mediaserver

编译命令： 在SDK根目录下，`make mediaserver-dirclean && make mediaserver`

10 libIPCProtocol

10.1 开发基础

基于dbus，提供进程间通信的函数接口。

开发语言： C

代码路径： app/LibIPCProtocol

编译命令： 在SDK根目录下，`make libIPCProtocol-dirclean && make libIPCProtocol`

10.2 对外接口

其中接口为对dbus通信的封装，各个服务的对外接口均在此库中提供。
核心都是通过dbus调用其他应用的method，但交互方式主要有两种：

第一种方式：通过dbus调用dbserver的method，将数据写入数据库，同时广播出去。关心此参数的应用可以通过监听，来进行实时处理。

优点：当有多个应用关心同一个参数时，不必调用多个应用的接口，而是让应用自己通过监听处理。且状态保存在数据库中，应用可以调用_get结尾的函数，来读取数据进行初始化。

缺点：应用需要增加监听dbus的部分。

样例如下：

```
char *dbserver_media_get(char *table);
/*
 * 传入数据库表名，char *table = "audio"
 * 返回值格式化后如下
 */
{
    "iReturn": 0,
    "sErrMsg": "",
    "jData": [
        {
            "id": 0,
            "sEncodeType": "AAC",
            "iSampleRate": 16000,
            "iBitRate": 32000,
            "sInput": "micIn",
            "iVolume": 50,
            "sANS": "close"
        } ]
}

char *dbserver_media_set(char *table, char *json, int id);
/*
 * 传入数据库表名，char *table = "audio"。
 * 传入数据库表索引，id = 0。
 * 传入数据，char *json = "{\"iVolume\":50}"，可以同时传多个参数，只要此表中有这些参数即可。
 * 其他监听此表数据变化的应用，将收到如下的dbus消息
 */
{
    "table": "audio",
```

```
    "key": {
        "id": 0
    },
    "data": {
        "iVolume": 50
    },
    "cmd": "Update"
}
```

第二种方式：通过dbus直接远程调用具体应用的method，大多数为不需要保存状态在数据库中的操作。如拍照、硬盘格式化、系统重启等。

优点：应用不需要进行监听，只需要提供远程调用的接口，节省代码量。

缺点：操作的状态无法保存。且如果某参数涉及多个应用，则每个修改者在修改参数的同时，还需要调用多个函数。

以系统重启为例，通过dbus相关函数，远程调用rockchip.system.server接口的Reboot方法。相应接口的应用收到消息后，就会执行对应此方法的函数。核心代码如下：

```
#define SYSTEM_MANAGER "rockchip.system"
#define SYSTEM_MANAGER_PATH "/"
#define SYSTEM_MANAGER_INTERFACE SYSTEM_MANAGER ".server"

dbus_method_call(userdata->connection,
                     SYSTEM_MANAGER, SYSTEM_MANAGER_PATH,
                     SYSTEM_MANAGER_INTERFACE, "Reboot",
                     populate_get, userdata, NULL, NULL);
```

10.3 注意事项

1. 由于返回字符串的长度不固定，所以某些函数中动态申请了内存，要注意内存释放问题。
2. mediaserver目前没有监听参数变化，所以音视频相关参数除了写入数据库外，还需要再调用mediaserver.h提供的接口进行配置。

11 ispserver

11.1 开发基础

图像信号处理服务端，具体开发请参考《ISP IPC模块框架说明及接口规范》

开发语言： C

代码路径： external/isp2-ipc

编译命令： 在SDK根目录下，`make isp2-ipc-dirclean && make isp2-ipc`

12 onvif_server

12.1 开发基础

onvif协议服务端。

开发语言： C

参考文档:

WSDL教程

SOAP教程

Web Services教程

onvif规范

代码路径： app/onvif_server

编译命令： 在SDK根目录下，`make onvif_server-dirclean && make onvif_server`

12.2 开发环境

1. 下载gSOAP工具包，并编译安装。
2. 根据onvif官网各个profile的要求，确定所需的wsdl文件。使用wsdl2h工具，wsdl文件，转换为纯C风格的头文件onvif.h。typemap.dat文件位于gSOAP工具包解压目录的gsoap文件夹下。

```
wsdl2h -c -s -t typemap.dat -o onvif.h
http://www.onvif.org/onvif/ver10/network/wsdl/remotediscovery.wsdl
http://www.onvif.org/onvif/ver10/device/wsdl/devicemgmt.wsdl
http://www.onvif.org/onvif/ver20/analytics/wsdl/analytics.wsdl
http://www.onvif.org/onvif/ver10/analyticsdevice.wsdl
http://www.onvif.org/onvif/ver10/media/wsdl/media.wsdl
http://www.onvif.org/onvif/ver20/media/wsdl/media.wsdl
http://www.onvif.org/onvif/ver10/deviceio.wsdl
http://www.onvif.org/onvif/ver10/display.wsdl
http://www.onvif.org/onvif/ver10/event/wsdl/event.wsdl
http://www.onvif.org/onvif/ver20/imaging/wsdl/imaging.wsdl
http://www.onvif.org/onvif/ver10/recording.wsdl
http://www.onvif.org/onvif/ver10/replay.wsdl
http://www.onvif.org/onvif/ver10/search.wsdl
http://www.onvif.org/onvif/ver10/receiver.wsdl
http://www.onvif.org/onvif/ver20/ptz/wsdl/ptz.wsdl
```

3. 使用soapcpp2工具，用onvif.h头文件生成服务端开发需要的.h和.c文件

```
soapcpp2 -s -2 onvif.h -x -I ../gsoap/import/ -I ../gsoap/
```

4. 选取其中需要的部分，移到app/onvif_server的目录下，注意不要覆盖已实现的函数。
5. 根据具体需求，实现server_operation.c中的函数。输入参数和输出参数的结构体已经有详细定义在soapStub.h中，按规范填充实现即可。

12.3 调试环境

1. 确保运行onvif_server的设备，和需要对接的NVR或个人电脑，位于同一局域网内。

2. 运行NVR的发现设备，或个人电脑上运行ONVIF Device Manager、ONVIF Device Test Tool等工具来发现设备，再进行调试操作。
3. 调试具体功能时，可观看打印log，判断是否调用了相应函数。
4. 如果使用ONVIF Device Test Tool或其他抓包工具，可看到以下样式的数据流。

Request:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:tds="http://www.onvif.org/ver10/device/wsdl"
xmlns:tt="http://www.onvif.org/ver10/schema">
  <soap:Body>
    <tds:GetDNS />
  </soap:Body>
</soap:Envelope>
```

Response:

```
HTTP/1.1 200 OK
Server: gSOAP/2.8
X-Frame-Options: SAMEORIGIN
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 2109
Connection: close


<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
    xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-encoding"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsdd="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:chan="http://schemas.microsoft.com/ws/2005/02/duplex"
    xmlns:wsa5="http://www.w3.org/2005/08/addressing"
    xmlns:xmime="http://tempuri.org/xmime.xsd"
    xmlns:xop="http://www.w3.org/2004/08/xop/include"
    xmlns:ns1="http://www.onvif.org/ver20/analytics/humanface"
    xmlns:ns2="http://www.onvif.org/ver20/analytics/humanbody"
    xmlns:tt="http://www.onvif.org/ver10/schema"
    xmlns:wsrfbf="http://docs.oasis-open.org/wsrf/bf-2"
    xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
    xmlns:wsrfr="http://docs.oasis-open.org/wsrf/r-2"
    xmlns:ns3="http://www.onvif.org/ver20/media/wsdl"
    xmlns:tad="http://www.onvif.org/ver10/analyticsdevice/wsdl"
    xmlns:tan="http://www.onvif.org/ver20/analytics/wsdl"
    xmlns:tdn="http://www.onvif.org/ver10/network/wsdl"
    xmlns:tds="http://www.onvif.org/ver10/device/wsdl"
    xmlns:tev="http://www.onvif.org/ver10/events/wsdl"
    xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
    xmlns:timg="http://www.onvif.org/ver20/imaging/wsdl"
    xmlns:tls="http://www.onvif.org/ver10/display/wsdl"
    xmlns:tmd="http://www.onvif.org/ver10/deviceIO/wsdl"
    xmlns:tptz="http://www.onvif.org/ver20/ptz/wsdl"
    xmlns:trc="http://www.onvif.org/ver10/recording/wsdl"
```

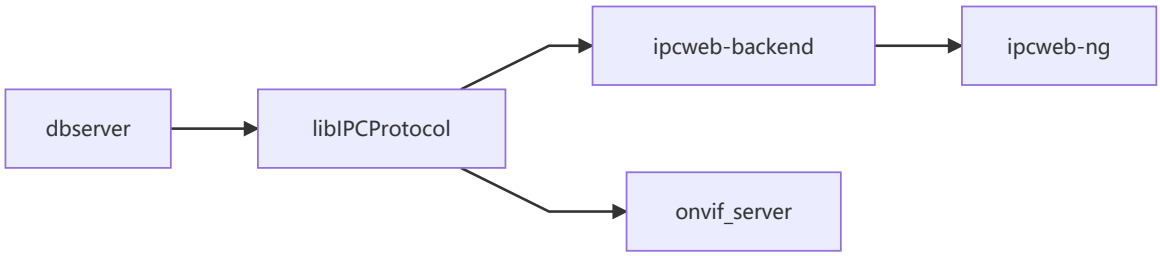
```
    xmlns:trp="http://www.onvif.org/ver10/replay/wsdl"
    xmlns:trt="http://www.onvif.org/ver10/media/wsdl"
    xmlns:trv="http://www.onvif.org/ver10/receiver/wsdl"
    xmlns:tse="http://www.onvif.org/ver10/search/wsdl">
    <SOAP-ENV:Body>
        <tds:GetDNSResponse>
            <tds:DNSInformation>
                <tt:FromDHCP>true</tt:FromDHCP>
                <tt:DNSFromDHCP>
                    <tt:Type>IPv4</tt:Type>
                    <tt:IPv4Address>10.10.10.188</tt:IPv4Address>
                </tt:DNSFromDHCP>
                <tt:DNSFromDHCP>
                    <tt:Type>IPv4</tt:Type>
                    <tt:IPv4Address>58.22.96.66</tt:IPv4Address>
                </tt:DNSFromDHCP>
            </tds:DNSInformation>
        </tds:GetDNSResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

12.4 注意事项

1. 所有指针都需要先直接或间接地调用soap_malloc进行申请内存。
2. 结构体在申请内存后，还需要调用soap_default_tt__开头的函数赋默认值，或手动给每一个成员赋值或NULL。注意不要遗漏，否则虽然编译能过，函数内部也不会报错，但是检查response结构体时的时候会直接退出，且难以排查。

13 应用框架开发流程

从数据库到web应用的开发，自底向上的开发流程如下：



1. dbserver：建表，并对数据进行初始化。
2. libIPCProtocol：封装对这张表进行读写操作的函数。调试可参考demo路径下的代码，编写测试程序，也可以用dbus-monitor工具，监控dbus总线。测试时，将编译生成的libIPCProtocol.so和测试程序推送到设备端即可。
3. ipcweb-backend：在相应的ApiHandler下，调用封装好的函数。用curl或postman测试对该URL进行get/put正常。测试时，将编译生成的entry.cgi推送到设备端即可。
4. ipcweb-ng：开发相应界面和注册回调，确保web前端可以正常地读写数据库。测试时，可在PC端直接指定URL中的IP地址为设备端IP地址进行调试，也可以将编译生成的文件夹推送到设备端，直接访问设备端IP地址进行调试。
5. onvif_server：封装符合onvif协议规范的接口函数，供其他符合onvif协议的客户端设备调用。

具体应用的开发，可以在以上第二步完成后并行开发。如果该操作无需保存状态，可以省去第一步。

应用主动获取配置信息有两种方式，一种是去读数据库的配置，另一种是监听相应的dbus接口。

前者通常用于进行初始化，可以在重启后读数据库的配置来进行初始化。

后者通常用于实时配置，当web端的命令转换为dbus总线上的消息时，可以一对多地广播。写入数据库保存配置的同时，也可以让监听到此消息的服务进行实时配置。

应用也可以提供dbus远程调用的接口，由上层来调用，而不主动获取配置信息。