**ft\_diag代码介绍**

1、什么是ft\_diag：是根据高通工具QXDM或QRCT发送的命令去执行某些我们想要的操作。简单来说，是将我们编写好的代码封装到命令集中去，通过命令去执行相应的代码。

2、路径：一般在vendor/byd/ft\_diag[Phoebe/LF4701]

3、代码分析：

1) bin目录下的代码：

Android.mk文件中链接了很多需要用到的文件，

libdiag\_includes:= \

$(LOCAL\_PATH)/../../qcom/proprietary/diag/include \

$(LOCAL\_PATH)/../../qcom/proprietary/diag/src 等等，基本上不用改动。

main.c 文件主要是将命令和函数连起来，比如：

DIAGPKT\_DISPATCH\_TABLE\_REGISTER\_V2(DIAG\_SUBSYS\_CMD\_VER\_2\_F, VERSION\_SUBSYS\_ID, version\_test\_tbl);

意思是将version\_test\_tbl数组函数与VERSION\_SUBSYS\_ID命令（VERSION\_SUBSYS\_ID是在factory\_test.h文件中定义的，值是0x01）连接起来，

version\_test\_tbl数组是在factory\_test.h中定义的，它包含很多的函数，

static const diagpkt\_user\_table\_entry\_type version\_test\_tbl[] =

{

{0x00, 0x00, PTAPI\_SW\_VER\_get\_internal},

{0x01, 0x01, PTAPI\_\_SW\_VER\_get\_external},

{0x02, 0x02, PTAPI\_HW\_VER\_get},

{0x03, 0x03, PTAPI\_HW\_VER\_set},

{0x04, 0x04, PTAPI\_TYPE\_DESIGNATOR\_get},

{0x05, 0x05, PTAPI\_VARIANT\_CODE\_get},

{0x06, 0x06, PTAPI\_VARIANT\_VER\_get}

};

如上代码，每个函数又与一个16进制对应，如PTAPI\_SW\_VER\_get\_internal对应0x00， PTAPI\_\_SW\_VER\_get\_external对应0x01，这样根据刚刚提到的VERSION\_SUBSYS\_ID和这个16进制就构成了diag的命令，比如PTAPI\_SW\_VER\_get\_internal函数对应的diag命令（都是16进制的）就是：

0x80 0x01 0x00 0x00 0x00 0x00 0x00 0x00：其中0x80是所有高通规定的diag命令的开头， 0x01就是VERSION\_SUBSYS\_ID值即DIAGPKT\_DISPATCH\_TABLE\_REGISTER\_V2(DIAG\_SUBSYS\_CMD\_VER\_2\_F, VERSION\_SUBSYS\_ID, version\_test\_tbl);函数定义的第二个参数，0x00是version\_test\_tbl数组定义的16进制，其余的补0x00，而PTAPI\_\_SW\_VER\_get\_external的diag命令就是0x80 0x01 0x01 0x00 0x00 0x00 0x00 0x00与PTAPI\_SW\_VER\_get\_internal的差别只是第三个16进制，只是version\_test\_tbl数组定义的16进制的不同，那么各个函数的具体实现是在factory\_test.c文件中，一般在factory\_test.c只是把返回结果显示到高通工具上，比如PTAPI\_SW\_VER\_get\_internal函数中调用了ft\_swVer\_get\_\_InternalVer函数，这个才是真正的实现函数，一般具体的实现函数都放在src目录下的文件夹下，ft\_swVer\_get\_\_InternalVer函数的实现是在version文件下的version.c文件中。

2)src目录：上面也提到了每个函数的具体实现代码都放在src目录下，并且在src目录下的Android.mk中添加需要编译的c文件，以LOCAL\_SRC\_FILES += 的方式。这里主要介绍下src目录下include目录里的内容，ft\_sys.h文件主要是定义一些常量，ft.h文件是定义src文件下的其他实现函数，在这里定义后才可以被bin目录下的factory\_test.c文件识别到，因为在factory\_test.c文件中#include "../src/include/ft.h"，所以factory\_test.c文件编译起来才不会报错说找不到函数。

4、举例说明：比如需要添加一个读版本号的功能[有读外部版本号和内部版本号]：

1）首先是在main.c文件中添加：

DIAGPKT\_DISPATCH\_TABLE\_REGISTER\_V2(DIAG\_SUBSYS\_CMD\_VER\_2\_F, VERSION\_SUBSYS\_ID, version\_test\_tbl);

2）在factory\_test.h文件中添加：#define VERSION\_SUBSYS\_ID 0x01

PACK(void\*)PTAPI\_SW\_VER\_get\_internal(PACK(void\*)req\_pkt, unsigned short pkt\_len);

PACK(void\*)PTAPI\_\_SW\_VER\_get\_external(PACK(void\*)req\_pkt, unsigned short pkt\_len);

//如果还有读取其他的版本，也可以继续在此添加其他函数的定义

static const diagpkt\_user\_table\_entry\_type version\_test\_tbl[] =

{

{0x00, 0x00, PTAPI\_SW\_VER\_get\_internal},

{0x01, 0x01, PTAPI\_\_SW\_VER\_get\_external},

//如果还有读取其他的版本，也可以继续在此添加其他函数和命令的链接

};

3）在factory\_test.c文件中添加，函数名和上面定义的函数名一致：

PACK(void\*)PTAPI\_SW\_VER\_get\_internal(PACK(void\*)req\_pkt, unsigned short pkt\_len)

{

int result = 0;

char log[80] = {0};

char version[80] = {0};

PACK(void\*)ft\_pkt = NULL;

ft\_pkt = diagpkt\_alloc((int)(\*(char\*)req\_pkt), sizeof(ft\_diag\_res\_pkt) + 60);

if (ft\_pkt == NULL)

{

return NULL;

}

memset(ft\_pkt, 0, sizeof(ft\_diag\_res\_pkt) + 60);

//result = property\_get( "ro.build.display.id.iversion", version, "" );

result = ft\_swVer\_get\_\_InternalVer(version);

if (result >= 0)

{

result = 1;

}

else

{

result = 0;

}

sprintf(log, "%d%s", result, version);

memcpy((void\*)ft\_pkt, (void\*)req\_pkt, pkt\_len);

strcpy(((ft\_diag\_res\_pkt\*)ft\_pkt)->my\_rsp, log);

diagpkt\_commit((PACK(void\*))ft\_pkt);

printf("diagpkt\_commit:%s\n", log);

return ft\_pkt;

}

PACK(void\*)PTAPI\_\_SW\_VER\_get\_external(PACK(void\*)req\_pkt, unsigned short pkt\_len)

{

int result = 0;

char log[80] = {0};

char version[80] = {0};

PACK(void\*)ft\_pkt = NULL;

ft\_pkt = diagpkt\_alloc((int)(\*(char\*)req\_pkt), sizeof(ft\_diag\_res\_pkt) + 60);

if (ft\_pkt == NULL)

{

return NULL;

}

memset(ft\_pkt, 0, sizeof(ft\_diag\_res\_pkt) + 60);

//result = property\_get( "ro.build.display.id.iversion", version, "" );

result = ft\_swVer\_get\_\_ExternalVer(version);

if (result >= 0)

{

result = 1;

}

else

{

result = 0;

}

sprintf(log, "%d%s", result, version);

memcpy((void\*)ft\_pkt, (void\*)req\_pkt, pkt\_len);

strcpy(((ft\_diag\_res\_pkt\*)ft\_pkt)->my\_rsp, log);

printf("diagpkt\_commit:%s\n", log);

diagpkt\_commit((PACK(void\*))ft\_pkt);

return ft\_pkt;

}

以上的函数中用到了具体的实现函数，ft\_swVer\_get\_\_InternalVer和ft\_swVer\_get\_\_ExternalVer。

4）在ft.h中添加上述函数的定义：int ft\_swVer\_get\_\_InternalVer(char \*version);

int ft\_swVer\_get\_\_ExternalVer(char \*version);

5）在src目录下新建version文件夹，在version文件夹中新建version.c，并实现以上2个函数的具体代码。

6）最后在src目录下的Android.mk中添加：LOCAL\_SRC\_FILES + = version.c

那么ft\_swVer\_get\_\_InternalVer和ft\_swVer\_get\_\_ExternalVer的diag命令就是：

#define VERSION\_SUBSYS\_ID 0x01

{0x00, 0x00, PTAPI\_SW\_VER\_get\_internal},

{0x01, 0x01, PTAPI\_\_SW\_VER\_get\_external},

===🡺

0x80 0x01 0x00 0x00 0x00 0x00 0x00 0x00

0x80 0x01 0x01 0x00 0x00 0x00 0x00 0x00

注意：

用QXDM工具是上面的16进制

用QRCT工具是去掉0x符号，即80 01 00 00 00 00 00 00，勾上16进制复选框

定义的命令不要与别的命令重复