# 1 音频基础

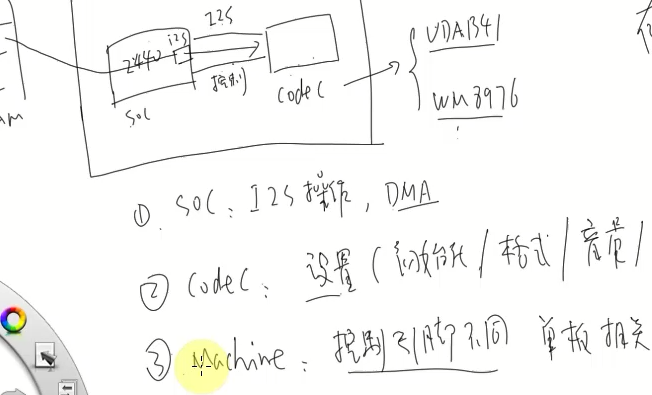
OSS和ALSA两种驱动框架，linux下目前使用ALSA.

ALSA包括软件框架和硬件操作

双通道数据记录是交替的，。

wav文件格式：

头部和数据两部分。



# 2 ALSA驱动框架分析(3.4.2内核)

sound/core/sound.c

与应用层接口

alsa\_sound\_init(void)

register\_chrdev(major, "alsa", &snd\_fops)

static const struct file\_operations snd\_fops = {

.owner = THIS\_MODULE,

.open = snd\_open,

.llseek = noop\_llseek,

};

snd\_open(struct inode \*inode, struct file \*file)//通用的open接口函数

mptr = snd\_minors[minor];

file->f\_op = fops\_get(mptr->f\_ops);

file->f\_op->open(inode, file);

何处注册了snd\_minors？

snd\_minors结构体定义如下

static struct snd\_minor \*snd\_minors[SNDRV\_OS\_MINORS];

从snd\_minors数组中取出一个结构体。该结构体在哪里注册呢？从static定义知道结构体在本文件中，高亮snd\_minors（F8）在本文件查找。找到注册地方

snd\_register\_device\_for\_dev（）

snd\_minors[minor] = preg;

struct snd\_minor {

int type; /\* SNDRV\_DEVICE\_TYPE\_XXX \*/

int card; /\* card number \*/

int device; /\* device number \*/

const struct file\_operations \*f\_ops; /\* file operations \*/

void \*private\_data; /\* private data for f\_ops->open \*/

struct device \*dev; /\* device for sysfs \*/

struct snd\_card \*card\_ptr; /\* assigned card instance \*/

};

snd\_register\_device\_for\_dev（）由谁调用？

一共有两处

sound/core.h里面的snd\_register\_device()创建声卡设备的控制接口

sound/core/pcm.c里面的snd\_pcm\_dev\_register(struct snd\_device \*device)

创建声卡设备的数据流接口

分析sound/core.h分支，创建声卡设备的控制接口

snd\_register\_device()由谁调用？

sound/core/control.c里面的函数snd\_ctl\_dev\_register（）调用

static int snd\_ctl\_dev\_register(struct snd\_device \*device)

snd\_register\_device（）

snd\_ctl\_dev\_register（）由snd\_ctl\_create（）函数设置

int snd\_ctl\_create(struct snd\_card \*card)

{

static struct snd\_device\_ops ops = {

.dev\_free = snd\_ctl\_dev\_free,

.dev\_register = snd\_ctl\_dev\_register,

.dev\_disconnect = snd\_ctl\_dev\_disconnect,

};

return snd\_device\_new(card, SNDRV\_DEV\_CONTROL, card, &ops);

}

snd\_ctl\_create由init.c (sound\core)里面的snd\_card\_create()函数调用

snd\_ctl\_create在特定的声卡文件里面调用

分析sound/core/pcm.c 创建声卡设备的数据流接口

static int snd\_pcm\_dev\_register(struct snd\_device \*device)

snd\_pcm\_dev\_register（）由\_snd\_pcm\_new（）调用

static int \_snd\_pcm\_new(struct snd\_card \*card, const char \*id, int device,

int playback\_count, int capture\_count, bool internal,

struct snd\_pcm \*\*rpcm)

🡪static struct snd\_device\_ops ops = {

.dev\_free = snd\_pcm\_dev\_free,

.dev\_register = snd\_pcm\_dev\_register,

.dev\_disconnect = snd\_pcm\_dev\_disconnect,

};

\_snd\_pcm\_new 由snd\_pcm\_new调用的

int snd\_pcm\_new(struct snd\_card \*card, const char \*id, int device,

int playback\_count, int capture\_count, struct snd\_pcm \*\*rpcm)

{

return \_snd\_pcm\_new(card, id, device, playback\_count, capture\_count,

false, rpcm);

}

snd\_pcm\_new 是由具体的声卡驱动调用的

由分析知道一个声卡要创建至少两种逻辑设备一个是控制设备，一个是数据流设备。

分别由snd\_ctl\_create（）和snd\_pcm\_new（）设置

分析snd\_ctl\_create流程

snd\_ctl\_create

.dev\_register = snd\_ctl\_dev\_register,

snd\_device\_new(card, SNDRV\_DEV\_CONTROL, card, &ops);

snd\_device\_new（）最终会导致snd\_ctl\_dev\_register被调用

snd\_ctl\_dev\_register(struct snd\_device\*device)

sprintf(name, "controlC%i", cardnum);//创建的控制设备的名字

snd\_register\_device(SNDRV\_DEVICE\_TYPE\_CONTROL, card, -1,

&snd\_ctl\_f\_ops, card, name))

int snd\_register\_device\_for\_dev(int type, struct snd\_card \*card, int dev,

const struct file\_operations \*f\_ops,

void \*private\_data,

const char \*name, struct device \*device)

preg = kmalloc(sizeof \*preg, GFP\_KERNEL);

preg->type = type;

preg->card = card ? card->number : -1;

preg->device = dev;

preg->f\_ops = f\_ops;

preg->private\_data = private\_data;

snd\_minors[minor] = preg;

preg->dev = device\_create(sound\_class, device, MKDEV(major, minor),

private\_data, "%s", name);

static const struct file\_operations snd\_ctl\_f\_ops =

{

.owner = THIS\_MODULE,

.read = snd\_ctl\_read,

.open = snd\_ctl\_open,

.release = snd\_ctl\_release,

.llseek = no\_llseek,

.poll = snd\_ctl\_poll,

.unlocked\_ioctl = snd\_ctl\_ioctl,

.compat\_ioctl = snd\_ctl\_ioctl\_compat,

.fasync = snd\_ctl\_fasync,

};

分析snd\_pcm\_new流程

int snd\_pcm\_new(struct snd\_card \*card, const char \*id, int device,

int playback\_count, int capture\_count, struct snd\_pcm \*\*rpcm)

\_snd\_pcm\_new(card, id, device, playback\_count, capture\_count,

false, rpcm);

.dev\_register = snd\_pcm\_dev\_register,

snd\_device\_new(card, SNDRV\_DEV\_PCM, pcm, &ops))

snd\_device\_new（）会导致snd\_pcm\_dev\_register（）被调用

snd\_pcm\_dev\_register（）

switch (cidx) {

case SNDRV\_PCM\_STREAM\_PLAYBACK:

//播放设备名字

sprintf(str, "pcmC%iD%ip", pcm->card->number, pcm->device);

devtype = SNDRV\_DEVICE\_TYPE\_PCM\_PLAYBACK;

break;

case SNDRV\_PCM\_STREAM\_CAPTURE:

//录音设备的名字

sprintf(str, "pcmC%iD%ic", pcm->card->number, pcm->device);

devtype = SNDRV\_DEVICE\_TYPE\_PCM\_CAPTURE;

break;

}

for (cidx = 0; cidx < 2; cidx++) //注册两个设备一个播放设备一个录音设备

snd\_register\_device\_for\_dev(devtype, pcm->card,

pcm->device,

&snd\_pcm\_f\_ops[cidx],

pcm, str, dev);

preg = kmalloc(sizeof \*preg, GFP\_KERNEL);

preg->type = type;

preg->card = card ? card->number : -1;

preg->device = dev;

preg->f\_ops = f\_ops;

preg->private\_data = private\_data;

snd\_minors[minor] = preg;

preg->dev = device\_create(sound\_class, device, MKDEV(major, minor),

private\_data, "%s", name);

const struct file\_operations snd\_pcm\_f\_ops[2] = {

{

.owner = THIS\_MODULE,

.write = snd\_pcm\_write,

.aio\_write = snd\_pcm\_aio\_write,

.open = snd\_pcm\_playback\_open,

.release = snd\_pcm\_release,

.llseek = no\_llseek,

.poll = snd\_pcm\_playback\_poll,

.unlocked\_ioctl = snd\_pcm\_playback\_ioctl,

.compat\_ioctl = snd\_pcm\_ioctl\_compat,

.mmap = snd\_pcm\_mmap,

.fasync = snd\_pcm\_fasync,

.get\_unmapped\_area = snd\_pcm\_get\_unmapped\_area,

},

{

.owner = THIS\_MODULE,

.read = snd\_pcm\_read,

.aio\_read = snd\_pcm\_aio\_read,

.open = snd\_pcm\_capture\_open,

.release = snd\_pcm\_release,

.llseek = no\_llseek,

.poll = snd\_pcm\_capture\_poll,

.unlocked\_ioctl = snd\_pcm\_capture\_ioctl,

.compat\_ioctl = snd\_pcm\_ioctl\_compat,

.mmap = snd\_pcm\_mmap,

.fasync = snd\_pcm\_fasync,

.get\_unmapped\_area = snd\_pcm\_get\_unmapped\_area,

}

};

何处创建设备类

根据device\_create(sound\_class, device, MKDEV(major, minor),private\_data, "%s", name);可知是在sound\_class类下面创建的设备，何处创建？

搜索sound\_class找到如下

init\_soundcore in sound\_core.c (sound) : \

sound\_class = class\_create(THIS\_MODULE, "sound");

init\_soundcore(void) //soundcore.c

sound\_class = class\_create(THIS\_MODULE, "sound");

创建的逻辑设备的名字和作用

|  |  |
| --- | --- |
| controlC0 -->  midiC0D0 -->  pcmC0D0c --〉  pcmC0D0p --〉  seq --〉  timer --〉 | 用于声卡的控制，例如通道选择，混音，麦克风的控制等 用于播放midi音频 用于录音的pcm设备 用于播放的pcm设备 音序器 定时器 |

怎么写ALSA声卡驱动

1. snd\_crad\_create
2. 初始化：创建逻辑设备

snd\_pcm\_new

1. snd\_card\_register

# 2 ASOC (ALSA system on chip)

这是声卡驱动的一个框架，分为三层

1. machine:单板相关的内容信息。抽象为snd\_soc\_card

表明platform是哪一个，cpu dai接口是哪个，DMA是哪一个

codec是哪一个 ，codec dai接口是哪一个

1. platform：

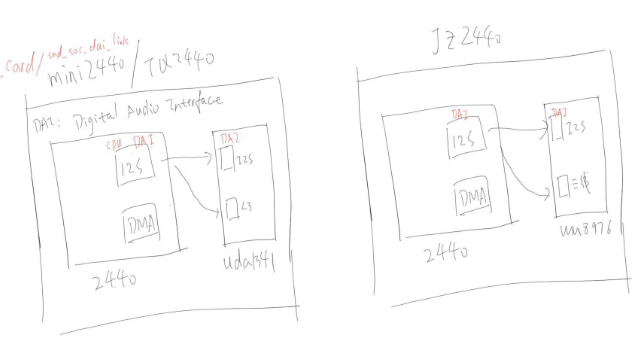
CPU DAI：初始化I2S,设置接口核心驱动是snd\_soc\_dai\_driver，设置i2s接口

DMA:对应一个snd\_soc\_plaform\_driver

1. codec

DAI :i2s snd\_soc\_dai\_driver

控制接口：对应一个snd\_soc\_codec\_driver



分析s3c2440的uda134x驱动

s3c24xx\_uda134x.c

定义一个平台设备驱动

static struct platform\_driver s3c24xx\_uda134x\_driver = {

.probe = s3c24xx\_uda134x\_probe,

.remove = s3c24xx\_uda134x\_remove,

.driver = {

.name = "s3c24xx\_uda134x",

.owner = THIS\_MODULE,

},

};

module\_platform\_driver(s3c24xx\_uda134x\_driver);//注册驱动

查找平台设备

搜索"s3c24xx\_uda134x"

在mach-mini2440.c中找到

static struct platform\_device mini2440\_audio = {

.name = "s3c24xx\_uda134x",

.id = 0,

.dev = {

.platform\_data = &mini2440\_audio\_pins,

},

};

s3c24xx\_uda134x.c

接下来分析probe函数

static struct snd\_soc\_ops s3c24xx\_uda134x\_ops = {

.startup = s3c24xx\_uda134x\_startup,

.shutdown = s3c24xx\_uda134x\_shutdown,

.hw\_params = s3c24xx\_uda134x\_hw\_params,

};

machine 驱动部分

s3c24xx\_uda134x.c

static struct snd\_soc\_dai\_link s3c24xx\_uda134x\_dai\_link = {

.name = "UDA134X",

.stream\_name = "UDA134X",

.codec\_name = "uda134x-codec",//用哪一个codec

.codec\_dai\_name = "uda134x-hifi",//codec芯片里哪一个DAI

.cpu\_dai\_name = "s3c24xx-iis",//2440的DAI接口

.ops = &s3c24xx\_uda134x\_ops,

.platform\_name = "samsung-audio",//DMA

};

static struct snd\_soc\_card snd\_soc\_s3c24xx\_uda134x = {

.name = "S3C24XX\_UDA134X",

.owner = THIS\_MODULE,

.dai\_link = &s3c24xx\_uda134x\_dai\_link,

.num\_links = 1,

};

static struct uda134x\_platform\_data s3c24xx\_uda134x = {

.l3 = {

.setdat = setdat,

.setclk = setclk,

.setmode = setmode,

.data\_hold = 1,

.data\_setup = 1,

.clock\_high = 1,

.mode\_hold = 1,

.mode = 1,

.mode\_setup = 1,

},

};

分析s3c24xx\_uda134x\_probe（）函数

//分配一个soc-audio设备，那么就会有对应的驱动，驱动和设备匹配就调用probe函数

s3c24xx\_uda134x\_snd\_device = platform\_device\_alloc("soc-audio", -1);

platform\_set\_drvdata(s3c24xx\_uda134x\_snd\_device,

&snd\_soc\_s3c24xx\_uda134x);

platform\_device\_add\_data(s3c24xx\_uda134x\_snd\_device, &s3c24xx\_uda134x, sizeof(s3c24xx\_uda134x));

ret = platform\_device\_add(s3c24xx\_uda134x\_snd\_device);

查找soc-audio定义

soc-core.c中找到如下和soc-audio相关的平台设备驱动

static struct platform\_driver soc\_driver = {

.driver = {

.name = "soc-audio",

.owner = THIS\_MODULE,

.pm = &snd\_soc\_pm\_ops,

},

.probe = soc\_probe,

.remove = soc\_remove,

};

static int soc\_probe(struct platform\_device \*pdev)

struct snd\_soc\_card \*card = platform\_get\_drvdata(pdev);

card->dev = &pdev->dev;

ret = snd\_soc\_register\_card(card);//在此处注册声卡，machine音频驱//动核心部分，声卡描述部分是前面的s3c24xx\_uda1314

抽象为snd\_soc\_card

开始分析platform驱动部分

CPU DAI

查找"s3c24xx-iis"，在s3c24xx-i2s.c找到CPU DAI,找到驱动

static struct platform\_driver s3c24xx\_iis\_driver = {

.probe = s3c24xx\_iis\_dev\_probe,

.remove = \_\_devexit\_p(s3c24xx\_iis\_dev\_remove),

.driver = {

.name = "s3c24xx-iis",

.owner = THIS\_MODULE,

},

};

struct platform\_device s3c\_device\_iis = {

.name = "s3c24xx-iis",

.id = -1,

.num\_resources = ARRAY\_SIZE(s3c\_iis\_resource),

.resource = s3c\_iis\_resource,

.dev = {

.dma\_mask = &samsung\_device\_dma\_mask,

.coherent\_dma\_mask = DMA\_BIT\_MASK(32),

}

};

s3c24xx\_iis\_dev\_probe

snd\_soc\_register\_dai(&pdev->dev, &s3c24xx\_i2s\_dai);

static struct snd\_soc\_dai\_driver s3c24xx\_i2s\_dai = {

.probe = s3c24xx\_i2s\_probe,

.suspend = s3c24xx\_i2s\_suspend,

.resume = s3c24xx\_i2s\_resume,

.playback = {

.channels\_min = 2,//支持的最小通道数

.channels\_max = 2, //支持的最大通道数

.rates = S3C24XX\_I2S\_RATES,//采样频率48KHZ

.formats = SNDRV\_PCM\_FMTBIT\_S8 | SNDRV\_PCM\_FMTBIT\_S16\_LE,},//数据格式8位还是16位

.capture = {

.channels\_min = 2,//

.channels\_max = 2,

.rates = S3C24XX\_I2S\_RATES,

.formats = SNDRV\_PCM\_FMTBIT\_S8 | SNDRV\_PCM\_FMTBIT\_S16\_LE,},

.ops = &s3c24xx\_i2s\_dai\_ops,

};

//iis接口支持的格式通道数等

static const struct snd\_soc\_dai\_ops s3c24xx\_i2s\_dai\_ops = {

.trigger = s3c24xx\_i2s\_trigger,

.hw\_params = s3c24xx\_i2s\_hw\_params,

.set\_fmt = s3c24xx\_i2s\_set\_fmt,

.set\_clkdiv = s3c24xx\_i2s\_set\_clkdiv,

.set\_sysclk = s3c24xx\_i2s\_set\_sysclk,

};

DMA

搜索"samsung-audio"//在machine处指定

struct platform\_device samsung\_asoc\_dma = {

.name = "samsung-audio",

.id = -1,

.dev = {

.dma\_mask = &samsung\_device\_dma\_mask,

.coherent\_dma\_mask = DMA\_BIT\_MASK(32),

}

};

在sound/soc/samsung/Dma.c里面找到

static struct platform\_driver asoc\_dma\_driver = {

.driver = {

.name = "samsung-audio",

.owner = THIS\_MODULE,

},

.probe = samsung\_asoc\_platform\_probe,

.remove = \_\_devexit\_p(samsung\_asoc\_platform\_remove),

};

static int \_\_devinit samsung\_asoc\_platform\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_platform(&pdev->dev, &samsung\_asoc\_platform);

// Register a platform with the ASoC core

static struct snd\_soc\_platform\_driver samsung\_asoc\_platform = {

.ops = &dma\_ops,

.pcm\_new = dma\_new,

.pcm\_free = dma\_free\_dma\_buffers,

};

static struct snd\_pcm\_ops dma\_ops = {

.open = dma\_open,

.close = dma\_close,

.ioctl = snd\_pcm\_lib\_ioctl,

.hw\_params = dma\_hw\_params,

.hw\_free = dma\_hw\_free,

.prepare = dma\_prepare,

.trigger = dma\_trigger,

.pointer = dma\_pointer,

.mmap = dma\_mmap,

};

codec部分分析

查找"uda134x-codec"

mach-mini2440.c (arch\arm\mach-s3c24xx) line 511 :

static struct platform\_device uda1340\_codec = {

.name = "uda134x-codec",

.id = -1,

};

uda134x.c (sound\soc\codecs) line 619 :

.driver = {

.name = "uda134x-codec",

.owner = THIS\_MODULE,

},

.probe = uda134x\_codec\_probe,

.remove = \_\_devexit\_p(uda134x\_codec\_remove),

};

static int \_\_devinit uda134x\_codec\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_codec(&pdev->dev,&soc\_codec\_dev\_uda134x, &uda134x\_dai, 1);

//完成读写uda1341寄存器的操作。控制接口

static struct snd\_soc\_codec\_driver soc\_codec\_dev\_uda134x = {

.probe = uda134x\_soc\_probe,

.remove = uda134x\_soc\_remove,

.suspend = uda134x\_soc\_suspend,

.resume = uda134x\_soc\_resume,

.reg\_cache\_size = sizeof(uda134x\_reg),

.reg\_word\_size = sizeof(u8),

.reg\_cache\_default = uda134x\_reg,

.reg\_cache\_step = 1,

.read = uda134x\_read\_reg\_cache,

.write = uda134x\_write,

.set\_bias\_level = uda134x\_set\_bias\_level,

};

//对应codec 数据接口

static struct snd\_soc\_dai\_driver uda134x\_dai = {

.name = "uda134x-hifi",

/\* playback capabilities \*/

.playback = {

.stream\_name = "Playback",

.channels\_min = 1,

.channels\_max = 2,

.rates = UDA134X\_RATES,

.formats = UDA134X\_FORMATS,

},

/\* capture capabilities \*/

.capture = {

.stream\_name = "Capture",

.channels\_min = 1,

.channels\_max = 2,

.rates = UDA134X\_RATES,

.formats = UDA134X\_FORMATS,

},

/\* pcm operations \*/

.ops = &uda134x\_dai\_ops,

};

static const struct snd\_soc\_dai\_ops uda134x\_dai\_ops = {

.startup = uda134x\_startup,

.shutdown = uda134x\_shutdown,

.hw\_params = uda134x\_hw\_params,

.digital\_mute = uda134x\_mute,

.set\_sysclk = uda134x\_set\_dai\_sysclk,

.set\_fmt = uda134x\_set\_dai\_fmt,

};

分析s3c24xx-i2s.c驱动

int s3c24xx\_iis\_dev\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_dai(&pdev->dev, &s3c24xx\_i2s\_dai);

struct snd\_soc\_dai \*dai;

dai->dev = dev;

dai->driver = dai\_drv;

list\_add(&dai->list, &dai\_list);

//把s3c24xx\_i2s\_dai放入dai\_list列表,.name = “s3c24xx-iis”

sound/soc/samsung/dma.c //把samsung\_asoc\_platform放入了platform\_list

samsung\_asoc\_platform\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_platform(&pdev->dev, &samsung\_asoc\_platform);

struct snd\_soc\_platform \*platform;

platform->dev = dev;

platform->driver = platform\_drv;

list\_add(&platform->list, &platform\_list);

codec:uda134x.c

uda134x\_codec\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_codec(&pdev->dev,

&soc\_codec\_dev\_uda134x, &uda134x\_dai, 1);

struct snd\_soc\_codec \*codec;

codec->dev = dev;

codec->driver = codec\_drv;

// codec\_drv= &soc\_codec\_dev\_uda134x

snd\_soc\_register\_dais(dev, dai\_drv, num\_dai);

//uda134x\_dai

list\_add(&codec->list, &codec\_list);

dai\_list🡪s3c24xx\_i2s\_dai🡪uda134x\_dai

platform\_list🡪samsung\_asoc\_platform

codec\_list🡪codec(driver=&soc\_codec\_dev\_udal34x)

machine部分驱动

s3c24xx\_uda134x.c

int s3c24xx\_uda134x\_probe(struct platform\_device \*pdev)

s3c24xx\_uda134x\_snd\_device = platform\_device\_alloc("soc-audio", -1);

platform\_set\_drvdata(s3c24xx\_uda134x\_snd\_device,

&snd\_soc\_s3c24xx\_uda134x);

platform\_device\_add\_data(s3c24xx\_uda134x\_snd\_device, &s3c24xx\_uda134x, sizeof(s3c24xx\_uda134x));

platform\_device\_add(s3c24xx\_uda134x\_snd\_device);

s3c24xx\_uda134x.c中注册了一个"soc-audio"设备

导致soc-core.c("soc-audio"的驱动)里面的soc-probe函数被调用

soc\_probe(struct platform\_device \*pdev)

snd\_soc\_register\_card(card);

card->rtd = devm\_kzalloc(card->dev,//运行时间

for (i = 0; i < card->num\_links; i++)

card->rtd[i].dai\_link = &card->dai\_link[i];

list\_add(&card->list, &card\_list);

snd\_soc\_instantiate\_cards();//实例化声卡

list\_for\_each\_entry(card, &card\_list, list)

snd\_soc\_instantiate\_card(card);

for (i = 0; i < card->num\_links; i++)

soc\_bind\_dai\_link(card, i);//soc的dai和codec的dai绑

/\* find CPU DAI f \*/

list\_for\_each\_entry(cpu\_dai, &dai\_list, list) {

rtd->cpu\_dai = cpu\_dai;//s3c24xx\_iis\_dai

/\* find CODEC \*/

rtd->codec = codec;

/\* CODEC found, so find CODEC DAI \*/

rtd->codec\_dai = codec\_dai;

/\* find one platforms \*/

rtd->platform = platform

/\* initialize the register cache for each available codec \*/

snd\_soc\_init\_codec\_cache(codec, compress\_type);

snd\_card\_create(SNDRV\_DEFAULT\_IDX1, SNDRV\_DEFAULT\_STR1,

card->owner, 0, &card->snd\_card);

soc\_probe\_dai\_link(card, i, order);

//调用cup\_dai的probe函数

cpu\_dai->driver->probe(cpu\_dai); /\* probe the cpu\_dai \*/

soc\_probe\_codec(card, codec); /\* probe the CODEC \*/

soc\_probe\_platform(card, platform); /\* probe the platform \*/

codec\_dai->driver->probe(codec\_dai); /\* probe the CODEC DAI \*/

soc\_new\_pcm(rtd, num);

snd\_card\_register(card->snd\_card);

# ALSA-lib 学习