专题14-串口驱动程序设计

一、tty驱动架构

1.1、TTY概念解析

1.1.1、/dev/ttySAC0

在Linux系统中，终端是一类字符型设备，它包括多 种类型，通常使用tty来简称各种类型的终端设备。

• 串口终端（/dev/ttyS*）

串口终端是使用计算机串口连接的终端设备。Linux 把每个串行端口都看作是一个字符设备。这些串行端口所对应的设备名称是 /dev/ttySAC0; /dev/ttySAC1……

1.1.2、/dev/tty1-n

虚拟终端（/dev/tty*）

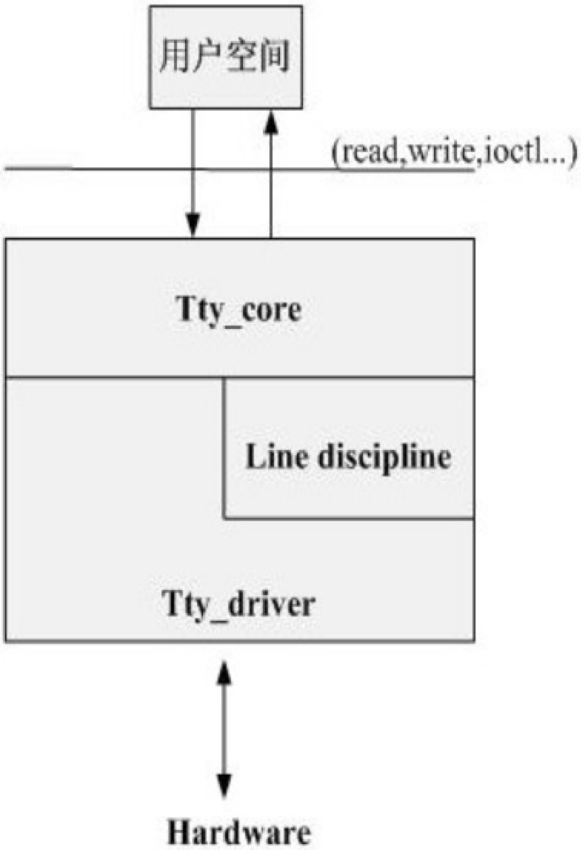
当用户登录时，使用的是虚拟终端。使用 Ctcl+Alt+[F1—F6]组合键时，我们就可以切换 到tty1、tty2、tty3等上面去。tty1–tty6等称为虚拟终端，而tty0则是当前所使用虚拟终端的 一个别名.

1.1.3、/dev/console

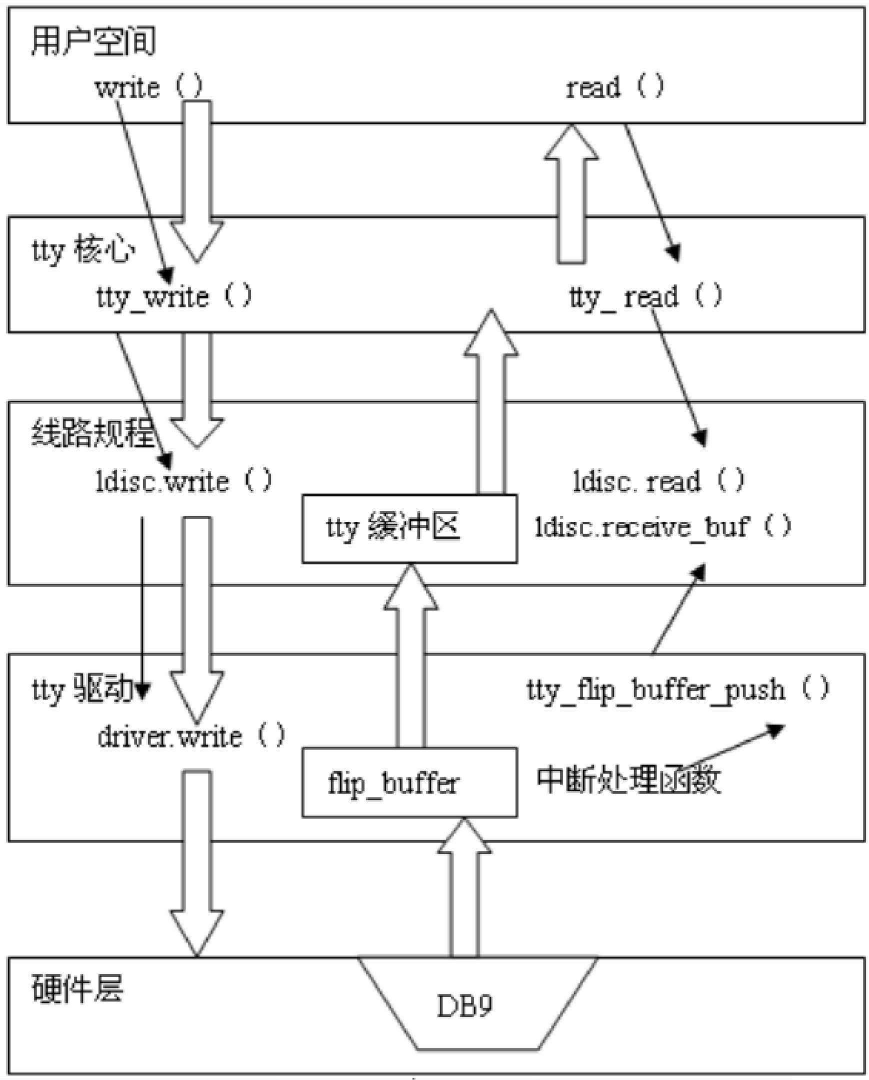
控制台终端（/dev/console）

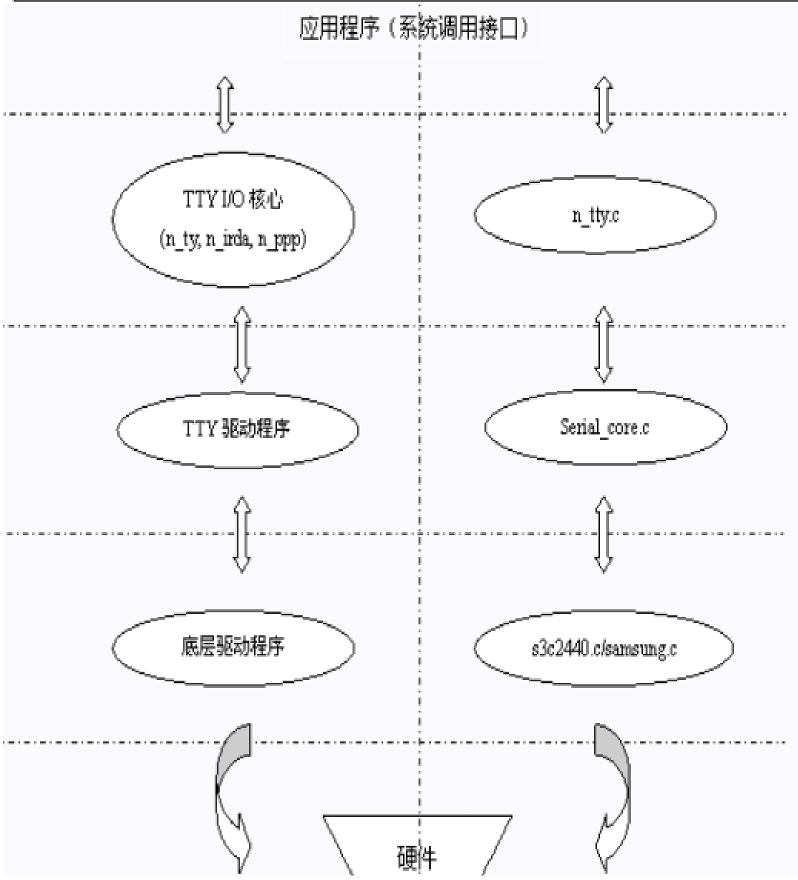
在Linux系统中，计算机的输出设备通常被称为控制 台终端(Console),这里特指printk信息输出到的设备。/dev/console是一个虚拟的设备，它需要映射到真 正的tty上，比如通过内核启动参数” console=ttySAC0”就把console映射到了串口0

1.2、TTY结构分析



Linux tty子系统包含：tty核 心，tty线路规程和tty驱动。 tty核心是对整个tty设备的抽 象，对用户提供统一的接口， tty线路规程是对传输数据的格式化，tty驱动则是面向tty 设备的硬件驱动。





1.3、回溯串口发送流程

dump_stack();用于显示调用信息。

```
[<c0049048>] (dump_backtrace+0x0/0x10c) from [<c037429c>] (dump_stack+0x18/0x1c)
 r7:c3942002 r6:00000000 r5:c38bda00 r4:c04caf20
[<c0374284>] (dump_stack+0x0/0x1c) from [<c018ea60>] (s3c24xx_serial_start_tx+0x14/0x64)
[<c018ea4c>] (s3c24xx_serial_start_tx+0x0/0x64) from [<c018ad78>] (uart_start+0x68/0x6c)
 r5:c38bda00 r4:60000013
[<c018ad10>] (uart_start+0x0/0x6c) from [<c018c5f8>] (uart_write+0xc0/0xe4)
 r5:c38bda00 r4:00000000
[<c018c538>] (uart_write+0x0/0xe4) from [<c0178034>] (n_tty_write+0x1d8/0x448)
[<c0177e5c>] (n_tty_write+0x0/0x448) from [<c0175790>] (tty_write+0x14c/0x244)
[<c0175644>] (tty_write+0x0/0x244) from [<c0175910>] (redirected_tty_write+0x88/0x98)
[<c0175888>] (redirected_tty_write+0x0/0x98) from [<c00a6e68>] (vfs_write+0xb4/0xe8)
 r9:c397e000 r8:c0045008 r7:00000004 r6:c397ff78 r5:40000000
r4:c3953700
[<c00a6db4>] (vfs_write+0x0/0xe8) from [<c00a6f80>] (sys_write+0x4c/0x84)
 r7:00000004 r6:c3953700 r5:00000000 r4:00000000
[<c00a6f34>] (sys_write+0x0/0x84) from [<c0044e60>] (ret_fast_syscall+0x0/0x2c)
 r6:001d27f8 r5:40000000 r4:00000002
```

tty核心：

```
ssize_t redirected_tty_write(struct file *file, const char __user *buf,
      size_t count, loff_t *ppos)
{
 struct file *p = NULL;

 spin_lock(&redirect_lock);
 if (redirect) {
  get_file(redirect);
  p = redirect;
 }
 spin_unlock(&redirect_lock);

 if (p) {
  ssize_t res;
  res = vfs_write(p, buf, count, &p->f_pos);
  fput(p);
  return res;
 }
 return tty_write(file, buf, count, ppos);
}

static ssize_t tty_write(struct file *file, const char __user *buf,
      size_t count, loff_t *ppos)
{
 struct inode *inode = file->f_path.dentry->d_inode;
 struct tty_struct *tty = file_tty(file);
  struct tty_ldisc *ld;
 ssize_t ret;

 if (tty_paranoia_check(tty, inode, "tty_write"))
  return -EIO;
 if (!tty || !tty->ops->write ||
  (test_bit(TTY_IO_ERROR, &tty->flags)))
   return -EIO;
 /* Short term debug to catch buggy drivers */
 if (tty->ops->write_room == NULL)
  printk(KERN_ERR "tty driver %s lacks a write_room method.\n",
   tty->driver->name);
 ld = tty_ldisc_ref_wait(tty);
 if (!ld->ops->write)
  ret = -EIO;
 else
  ret = do_tty_write(ld->ops->write, tty, file, buf, count);
 tty_ldisc_deref(ld);
 return ret;
}
```

线路规程：

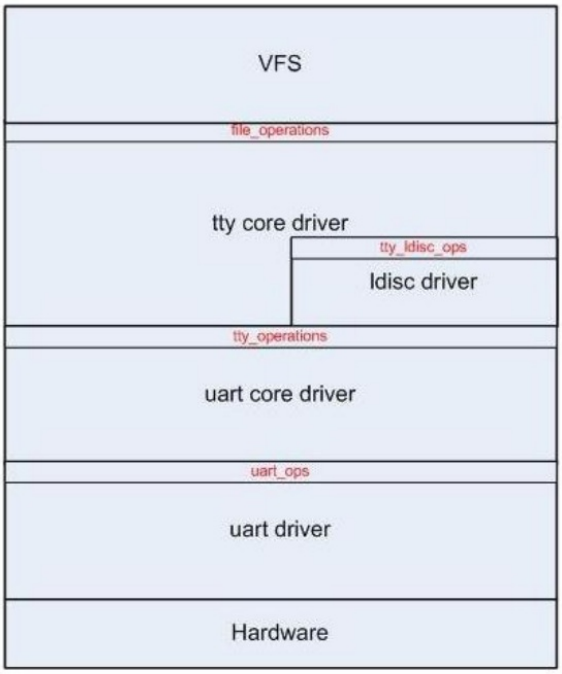
```
static ssize_t n_tty_write(struct tty_struct *tty, struct file *file,
      const unsigned char *buf, size_t nr)
```

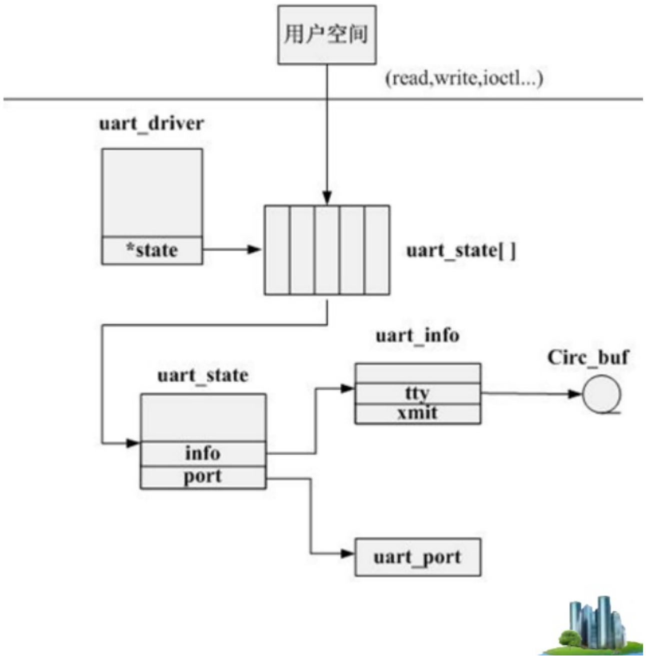
```
{
 const unsigned char *b = buf;
 DECLARE_WAITQUEUE(wait, current);
 int c;
 ssize_t retval = 0;

 /* Job control check -- must be done at start (POSIX.1 7.1.1.4). */
 if (L_TOSTOP(tty) && file->f_op->write != redirected_tty_write) {
  retval = tty_check_change(tty);
  if (retval)
   return retval;
 }

 /* Write out any echoed characters that are still pending */
 process_echoes(tty);

 add_wait_queue(&tty->write_wait, &wait);
 while (1) {
  set_current_state(TASK_INTERRUPTIBLE);
  if (signal_pending(current)) {
   retval = -ERESTARTSYS;
   break;
  }
  if (tty_hung_up_p(file) || (tty->link && !tty->link->count)) {
   retval = -EIO;
   break;
  }
  if (O_OPOST(tty) && !(test_bit(TTY_HW_COOK_OUT, &tty->flags))) {
   while (nr > 0) {
    ssize_t num = process_output_block(tty, b, nr);
    if (num < 0) {
     if (num == -EAGAIN)
      break;
     retval = num;
     goto break_out;
    }
    b += num;
    nr -= num;
    if (nr == 0)
     break;
    c = *b;
    if (process_output(c, tty) < 0)
     break;
    b++; nr--;
   }
   if (tty->ops->flush_chars)
    tty->ops->flush_chars(tty);
  } else {
   while (nr > 0) {
    c = tty->ops->write(tty, b, nr);
    if (c < 0) {
     retval = c;
     goto break_out;
    }
    if (!c)
     break;
    b += c;
    nr -= c;
   }
  }
  if (!nr)
   break;
  if (file->f_flags & O_NONBLOCK) {
   retval = -EAGAIN;
   break;
  }
  schedule();
 }
```

```
break_out:
 __set_current_state(TASK_RUNNING);
 remove_wait_queue(&tty->write_wait, &wait);
 if (b - buf != nr && tty->fasync)
  set_bit(TTY_DO_WRITE_WAKEUP, &tty->flags);
 return (b - buf) ? b - buf : retval;
}
```

二、串口驱动分析-初始化

2.1、串口驱动程序结构





2.2、串口驱动中的重要数据结构

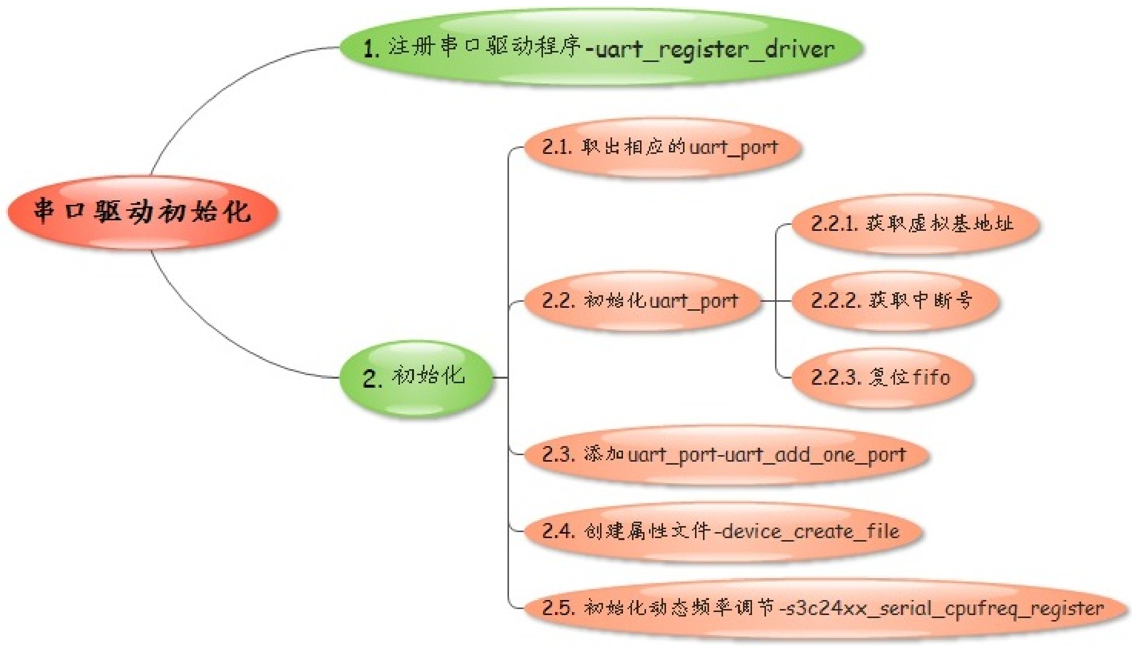
- UART驱动程序结构：struct uart_driver
- UART端口结构： struct uart_port
- UART相关操作函数结构: struct uart_ops
- UART状态结构： struct uart_state
- UART信息结构: struct uart_info

2.3、初始化分析

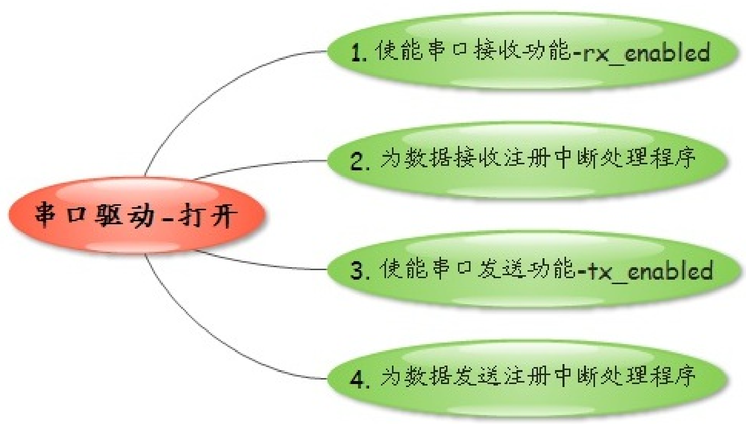
samsung.c是2440,S3C6410,210通用的文件，s3c6400.c是s3c6410处理器的文件，这两个文件合起来就是串口文件。

```
static int uart_write(struct tty_struct *tty,
     const unsigned char *buf, int count)
{
 struct uart_state *state = tty->driver_data;
 struct uart_port *port;
 struct circ_buf *circ;
 unsigned long flags;
 int c, ret = 0;

 /*
  * This means you called this function _after_ the port was
  * closed.  No cookie for you.
  */
 if (!state) {
  WARN_ON(1);
  return -EL3HLT;
 }

 port = state->uart_port;
 circ = &state->xmit;

 if (!circ->buf)
  return 0;
```

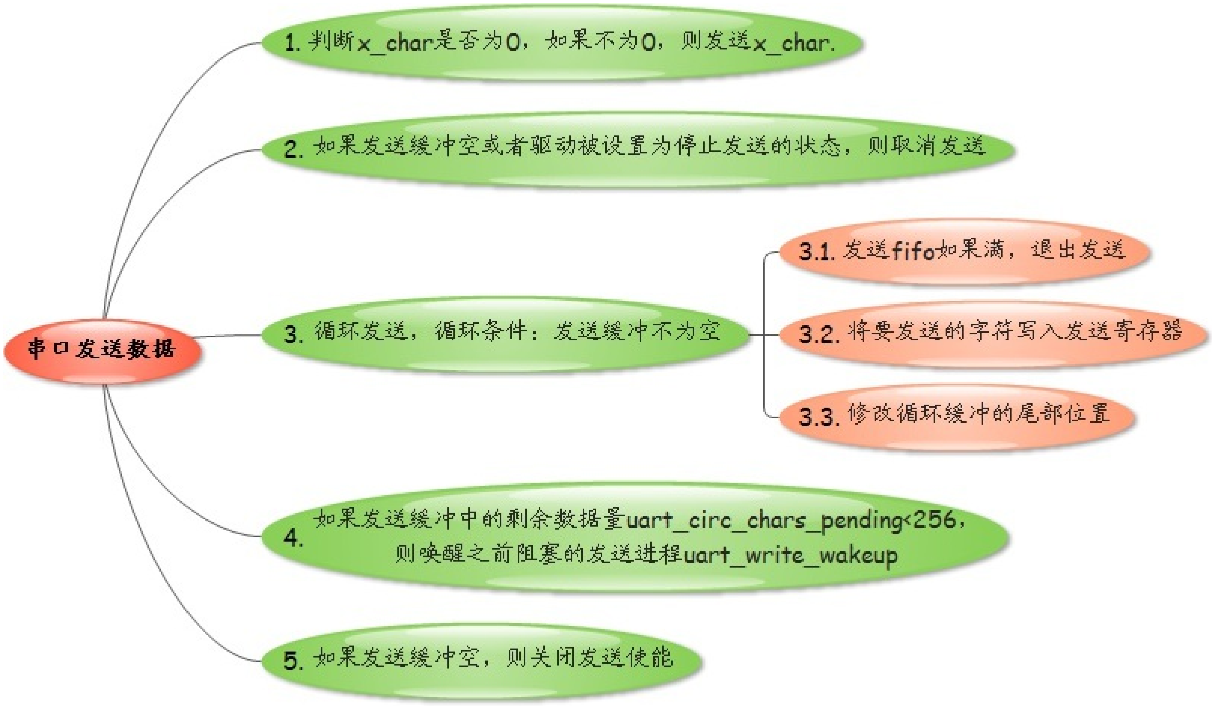
```
spin_lock_irqsave(&port->lock, flags);
while (1) {
 c = CIRC_SPACE_TO_END(circ->head, circ->tail, UART_XMIT_SIZE);
 if (count < c)
  c = count;
 if (c <= 0)
  break;
 memcpy(circ->buf + circ->head, buf, c);
 circ->head = (circ->head + c) & (UART_XMIT_SIZE - 1);
 buf += c;
 count -= c;
 ret += c;
}
spin_unlock_irqrestore(&port->lock, flags);

uart_start(tty);
return ret;
}
```



三、串口驱动分析-打开设备



四、串口驱动分析-数据发送



4.1、数据传递-循环缓冲

五、串口驱动分析-数据接收

5.1、流控

硬流控的RTS、CTS：

（现在做串口使用RTS/CTS必看内容，因为MTK/）

RTS （Require ToSend，发送请求）为输出信号，用于指示本设备准备好可接收数据，低电平有效，低电平说明本设备可以接收数据。

CTS （Clear ToSend，发送允许）为输入信号，用于判断是否可以向对方发送数据，低电平有效，低电平说明本设备可以向对方发送数据。

六、串口驱动编程实现

6.1、分析架构-子系统

6.2、分析驱动-流程图

6.3、实现代码