嵌入式系统原理

Principle of Embedded System

计算机与信息学院　　分布智能与物联网研究所

卫 星、石雷、毕翔

2022年5月

本章主要讲述

7.1 SHELL编程

7.2 BootLoader过程

7.3 各章复习概要

Shell程序的特点及用途

- shell程序可以认为是将shell命令按照控制结构组织到一个文本文件中，批量的交给shell去执行
- 不同的shell解释器使用不同的shell命令语法
- shell程序解释执行，不生成可以执行的二进制文件
- 可以帮助用户完成特定的任务，提高使用、维护系统的效率
- 了解shell程序可以更好的配置和使用linux

Shell程序举例

greeting.sh		解释
1	#!/bin/bash	以 #! 开始，其后为使用的shell
2	#a Simple shell Script Example	以 # 开始，其后为程序注释
3	#a Function	同上
4	function say_hello()	以 functin 开始，定义函数
5	{	函数开始
6	echo "Enter Your Name,Please. :"	echo命令输出字符串
7	read name	读入用户的输入到变量name
8	echo "Hello $name"	输出
9	}	函数结束
10	echo "Programme Starts Here....."	程序开始的第一条命令，输出提示信息
11	say_hello	调用函数
12	echo "Programme Ends."	输出提示，提示程序结束

程序编译和运行过程

- 一般步骤：
  - 编辑文件
  - 保存文件
  - 将文件赋予可以执行的权限
  - 运行及排错
- 常用到的命令：
  - vi，编辑、保存文件
  - ls -l 查看文件权限
  - chmod 改变程序执行权限
  - 直接键入文件名运行文件

Shell程序过程

学院 荣誉 责任

```
[tom@localhost ~]$ ll
总用量 20
-rw-r--r--  1 root root    0
drwxr-xr-x  2 tom  tom  4096
-rw-rw-r--  1 tom  tom   210  6月 29 22:58 greeting.sh
[tom@localhost ~]$ chmod +x greeting.sh
[tom@localhost ~]$ ll
总用量 20
-rw-r--r--  1 root root    0  6月 20 19:32 abc.txt
drwxr-xr-x  2 tom  tom  4096  6月 19 01:23 Desktop
-rwxrwxr-x  1 tom  tom   210  6月 29 22:58 greeting.sh
[tom@localhost ~]$ ./greeting.sh
Programme Starts Here.....
Enter Your Name,Please.   :
tom
Hello tom
Programme Ends.
[tom@localhost ~]$
```

查看权限

查看权限，初始状态无执行（x）权限

增加可执行（x）的权限

查看权限，已经具备执行（x）权限

运行程序

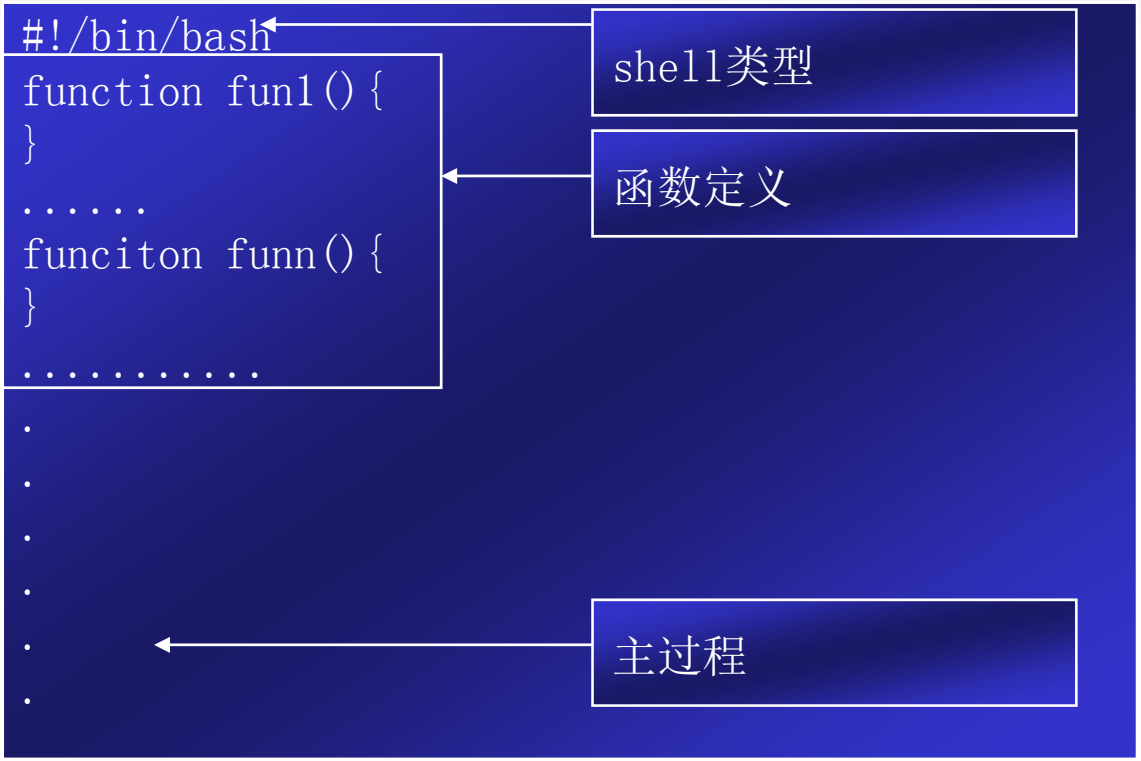
程序运行过程输出

另外两种运行调试shell脚本的方法

- 1. 赋予脚本文件可执行权限
- 2. bash<脚本文件名
- 3.bash 脚本文件名 参数

一般结构

- shell类型
- 函数
- 主过程



- Shell编程中，使用变量无需事先声明
- 变量的赋值与引用：
  - ✓ 赋值：变量名=变量值（注意：不能留空格）
  - ✓ 引用：$var引用var变量
- Shell变量有几种类型
  - ✓ 用户自定义变量
  - ✓ 环境变量
  - ✓ 位置参数变量
  - ✓ 专用参数变量

用户自定义变量

- 由用户自己定义、修改和使用，Shell的默认赋值是字符串赋值

```
var=1
var=$var+1
echo  $var              #打印的结果是什么呢？
```

- 为了达到想要的效果有以下几种表达方式
  - ✓ let "var+=1"
  - ✓ var=$[$var+1]
  - ✓ var=`expr $var + 1`#注意加号两边的空格

变量的引用

- **格式:**

  $变量名，或者${变量名}

  变量名为一个字符用方式一，变量名多于一个字符建议用第2种方式

- **例子：**

```
a=1
abc="hello"
echo $a
echo ${abc}
```

用户自定义变量

- $ aa=Hello
- $echo $aa
- Hello

- $ aa= “Yes Sir”
- $echo $aa
- Yes Sir

- $ aa=7+5
- $echo $aa
- 7+5

注：等号两边不能有空格，如果字符串两边有空格，必须加用引号

Shell常用内置命令

- echo：在屏幕上显示出由arg指定的字串
  - 命令格式：**echo arg**
- export
  - 命令格式：**export变量[=变量值]**
  - **Shell可以用export把它的变量向下带入子 Shell，从而让子进程继承父进程中的环境变量**
  - **不带任何变量名的export语句将显示出当前所 有的export变量**

Shell常用内置命令

- read：从键盘输入内容为变量赋值
  - **read [-p "信息"] [var1 var2 ...]**
    - 若省略变量名，则将输入的内容存入**REPLY**变量
- readonly：不能被清除或重新赋值的变量
  - **readonly variable**

双引号

- 双引号内的字符，除$、`和\仍保留其特殊功能外，其余字符均作为普通字符对待
- $表示变量替换
- 倒引号表示命令替换
- \为转义字符

- **echo “Dir is `pwd` and logname is $LOGNAME”**
- **names="Zhangsan Lisi Wangwu"**

■ 单引号

由单引号括起来的字符都作为普通字符出现， 即使是$、`和\

echo 'The time is `date` , the file is $HOME/abc '

The time is ` date ` , the file is $HOME/abc

■ 倒引号

倒引号括起来的字符串被shell解释为命令行， 在执行时，Shell会先执行该命令行，并以它的 标准输出结果取代整个倒引号部分。在前面示例中已经见过。

- **echo “Dir is `pwd` and logname is $LOGNAME”**
- **names="Zhangsan Lisi Wangwu"**

- **$[]**：可以接受不同基数的数字的表达式

  echo $[10+1]（输出：**11**）

  echo “$[2+3],$HOME”（输出：**5,/root**）

  echo $[2<<3],$[8>>1]（输出：**16,4**）

  echo $[2>3],$[3>2]（输出：**0,1** 表达式为**false**时输出**0**，为**true**时输出**1**）

- 字符表达式：直接书写，采用单引号，双引号引起来。

  echo " $HOME，That is your root directory."（输出：**/root，That is your root directory.**）

  echo ' $HOME，That is your root directory. '**（输出：**$HOME，That is your root directory.）

  单引号和双引号的区别在于：单引号是原样显示，双引号则显示出变量的值。

控制结构

根据某个条件的判断结果，改变程序执行的路径。可以简单的将控制结构分为分支和循环两种 。

- 常见分支结构：
  - **if**
  - **case**
- 常见循环结构：
  - **for**
  - **while**
  - **until**

- **if分支**
  - 格式：

```
if 条件1
then
命令
[elif 条件2
    then
命令]
[else
命令]
fi
```

- 说明：
  - 中括号中的部分可省略；
  - 当条件为真（0）时执行then后面的语句，否则执行else后面的语句；
  - 以fi作为if结构的结束。

- if分支

- #!/bin/bash

  #if.sh

  if ["10" -lt "12"] #注意：if和[之间，[和"10"之间，"12"和]都有空格，如果不加空格，会出现语法错误

  then

  echo "Yes,10 is less than 12"

  fi

- **case分支**
  - 格式：

```
case 条件 in
模式1）
        命令1
        ；；
[模式2）
        命令2
        ；；
.....................
模式n）
        命令n
        ；；]
esac
```

  - 说明：
    - “条件”可以是变量、表达式、shell命令等；
    - “模式”为条件的值，并且一个“模式”可以匹配多种值，不同值之间用竖线（|）联结；
    - 一个模式要用双分号（；；）作为结束；
    - 以逆序的case命令（esac）表示case分支语句的结束

```
#!/bin/bash
#case.sh
echo -n "Enter a start or stop:"
read ANS
case $ANS in
start)
echo "You select start"
;;
stop)
echo "You select stop"
;;
*)
echo "`basename $0`: You select is not between start and stop"
>&2
#注意：>和&2之间没有空格,>&2 表示将显示输出到标准输出（一般是屏幕）上
exit;
;;
esac
```

- **for循环**
- 格式

```
for 变量 [in 列表]
do
        命令（通常用到循环变量）
done
```

– 说明：

- **“列表”为存储了一系列值的列表，随着循环的进行，变量从列表中的第一个值依次取到最后一个值；**
- **do和done之间的命令通常为根据变量进行处理的一系列命令，这些命令每次循环都执行一次；**
- **如果中括号中的部分省略掉，Bash则认为是“in $@”，即执行该程序时通过命令行传给程序的所有参数的列表。**

```
#!/bin/sh

for foo in bar fud 43
do
  echo $foo
done
exit 0
```

That results in the following output:

```
bar
fud
43
```

- **while循环与until循环**
  - 格式：

```
while/until   条件
do
        命令
done
```

- 说明：
  - **while循环中，只要条件为真，就执行do和done之间的循环命令；**
  - **until循环中，只要条件不为真，就执行do和done之间的循环命令，或者说，在until循环中，一直执行do和done之间的循环命令，直到条件为真；**
  - 避免生成死循环。

```
#!/bin/sh

echo "Enter password"
read trythis

while [ "$trythis" != "secret" ]; do
  echo "Sorry, try again"
  read trythis
done
exit 0
```

An example of the output from this script is as follows:

```
Enter password
password
Sorry, try again
secret
$
```

- 函数
- 格式：
  - 定义：

```
[function] 函数名（）
{
        命令
}
```

  - 引用：

```
函数名 [ 参数1 参数2 ...参数n ]
```

- 说明：
  - 中括号中的部分可以省略；
  - 如果在函数内部需要使用传递给函数的参数，一般用**$0、$1、.....、$n，以及$#、$*、$@这些特殊变量：**
    - **$0为执行脚本的文件名；**
    - **$1是传递给函数的第1个参数；**
    - **$#为传递给函数的参数个数；**
    - **$*和$@为传递给函数的所有参数**

```
#!/bin/sh

foo() {
    echo "Function foo is executing"
}

echo "script starting"
foo
echo "script ended"

exit 0
```

Running the script will output the following:

```
script starting
Function foo is executing
script ending
```

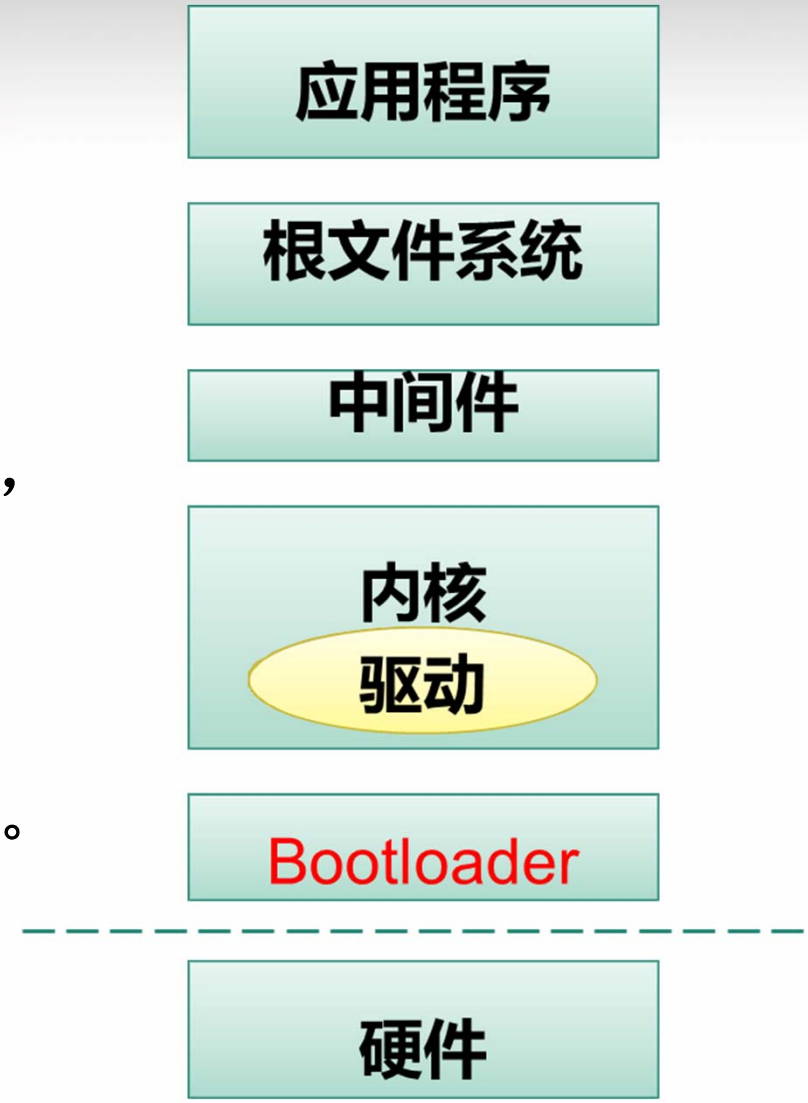
本章主要讲述

7.1 SHELL编程

7.2 BootLoader过程

7.3 各章复习概要

- **Bootloader** 就是在操作系统内核运行之前运行的 一段小程序。
- 通过这段小程序，可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。



- **Boot Loader**是严重地依赖于硬件而实现的， 包括**CPU**、嵌入式板级设备的配置等。
- 依赖于处理器架构： **ARM**、**MIPS**、**DSP**、**x86 etc**
- 依赖于具体的板级配置 ：不同厂家的芯片、不同的内存空间

下载模式(Downloading)

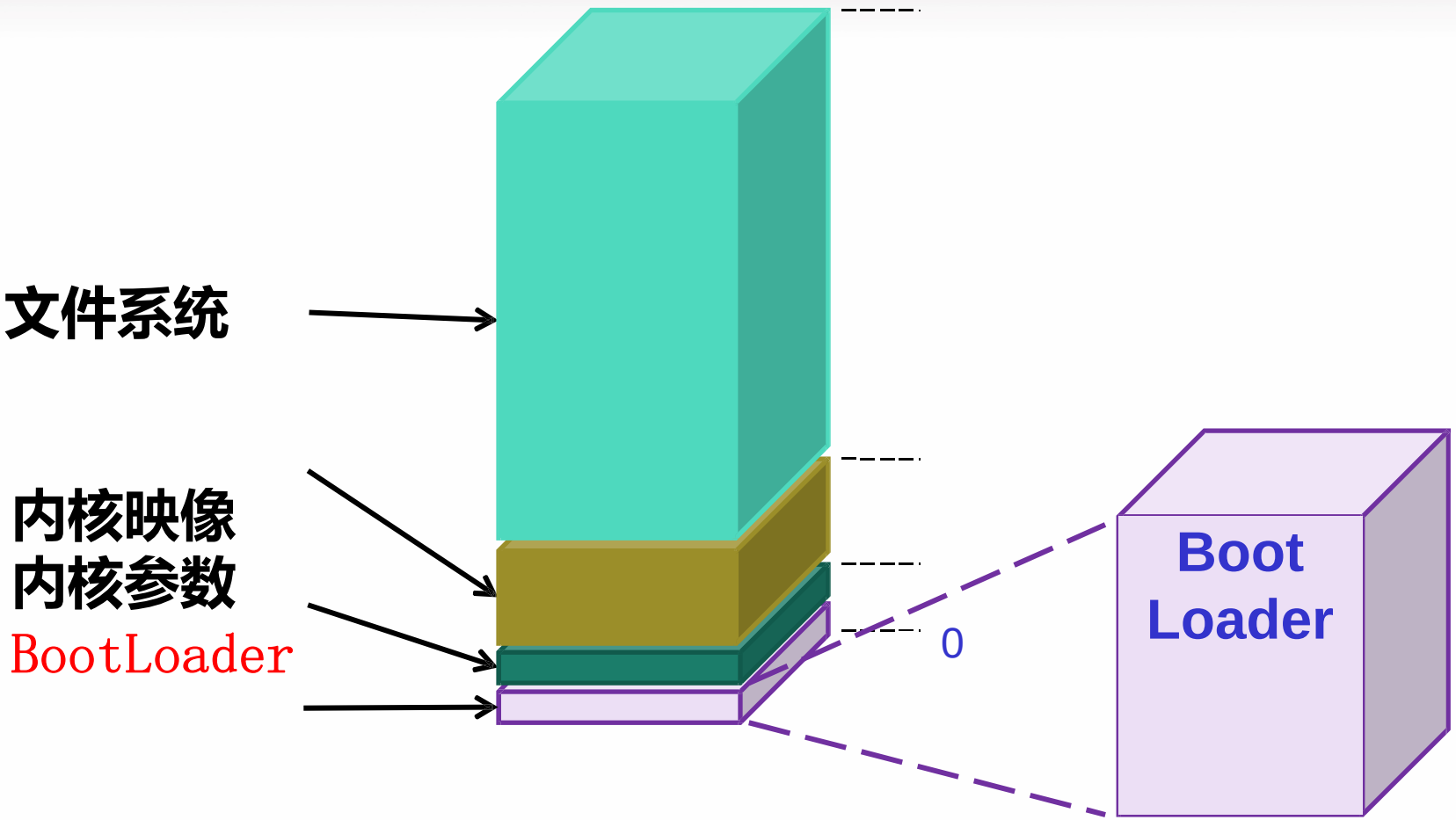
- 用于调试和版本修改：目标机上的 **Boot Loader** 将通过串口连接或网络连接等通信手段从主机下载文件到目标机的 **RAM** 中，然后再被 **Boot Loader** 写到目标机上的**FLASH** 类固态存储设备中
- 所谓的“刷机”就是使用这种模式
- 工作于这种模式下的 **Boot Loader**需要提供 一个简单的命令行接口

  常用命令**tftp**、**update**

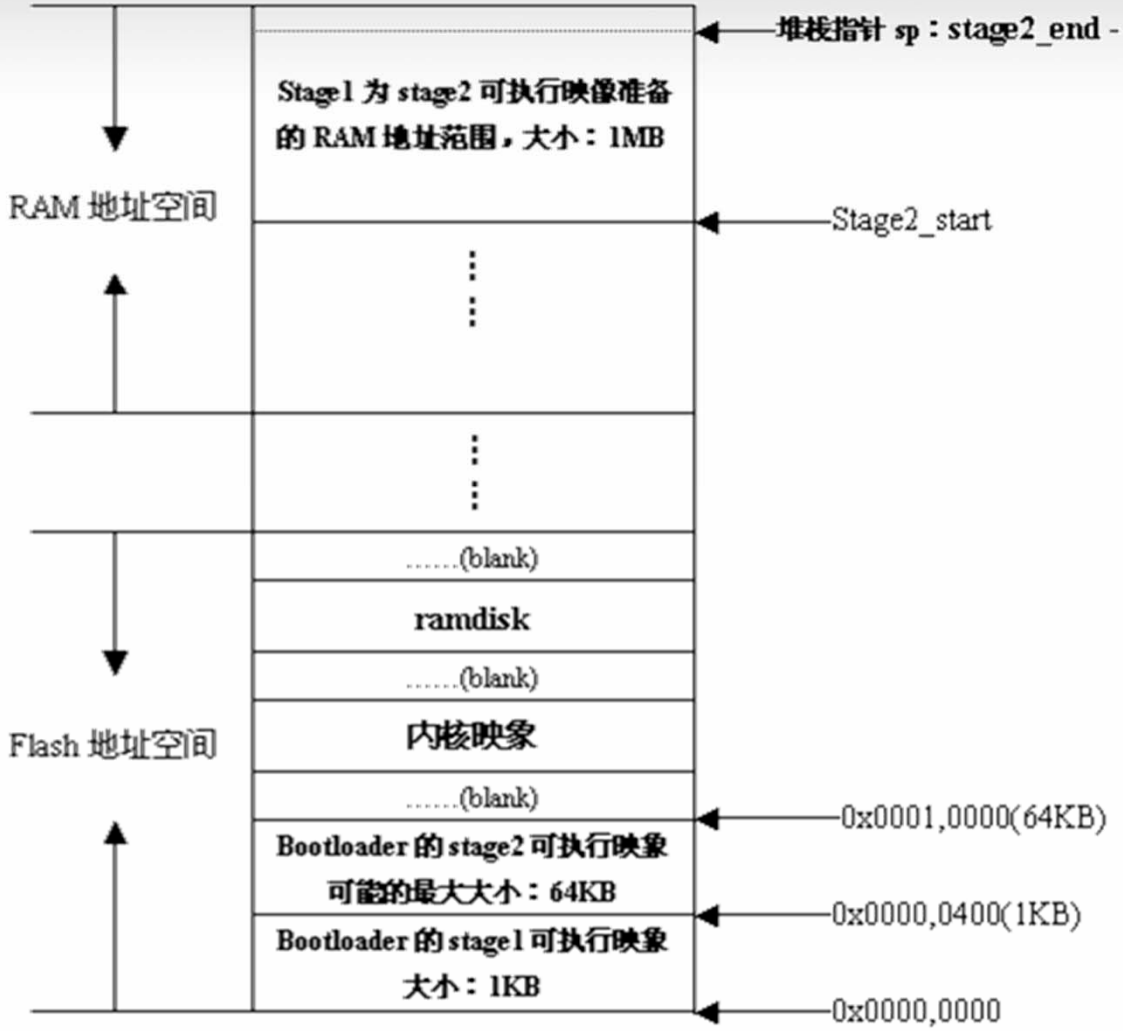
启动加载模式(Boot Loading)

- 自主（**Autonomous**）模式，是**BootLoader** 的正常工作模式
- 流程：
  - 从目标机某个固态存储设备上将**OS**加载到 **RAM**
  - 准备好内核运行所需的环境和参数
  - 在**RAM**运行操作系统内核

- 固态存储设备的典型空间分配结构



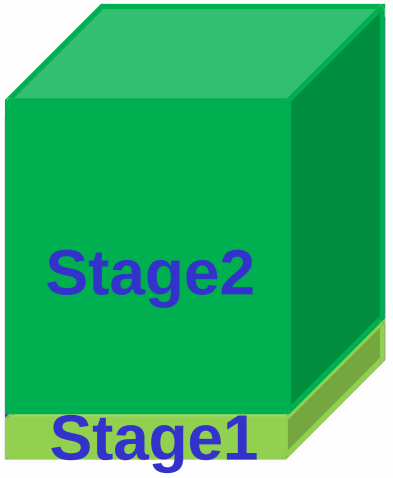
Bootloader 运行过程



BootLoader通常分为 stage1 和 stage2 两大部分

- Stage1——依赖于CPU体系结构的代码，例如设备初始化代码等。通常都用汇编语言来实现， 以达到短小精悍的目的
- Stage2——通常用C语言来实现，可以实现更复杂的功能，而且代码会具有更好的可读性和可移植性

分级载入机制



Stage1具体过程

1. 基本的硬件初始化，目的是为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境

(1) 屏蔽所有的中断

- 在 BootLoader 的执行全过程中可以不必响应任何中断
- 中断屏蔽通过写 CPU 的中断屏蔽寄存器或状态寄存器来完成

(2) 设置CPU的速度和时钟频率

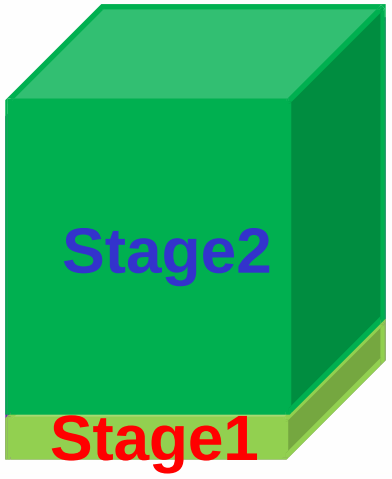
(3) RAM初始化

- 设置内存控制器的功能寄存器及各内存控制寄存器等

(4)初始化 LED

- 目的是表明系统的状态是 OK 还是 Error

(5)关闭 CPU 内部指令 / 数据 cache

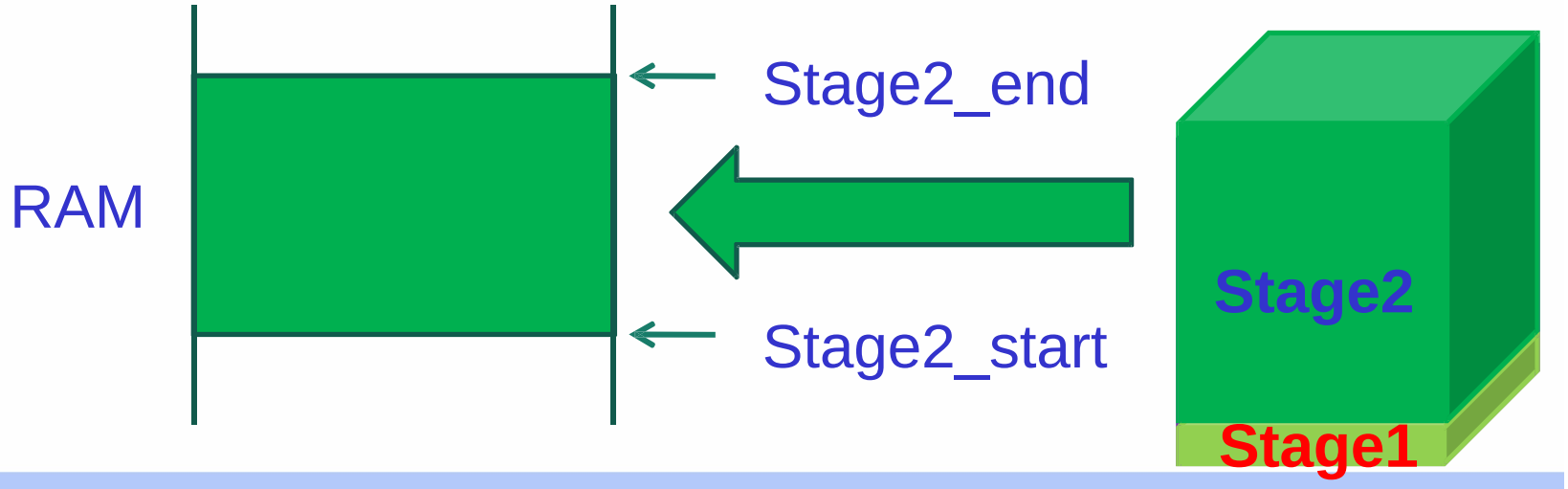


Stage1具体过程

2.为加载 stage2 准备 RAM 空间

为了获得更快的执行速度，通常把 stage2 加载到 RAM 空间中来执行，必须为加载 Boot Loader 的 stage2 准备好一 段可用的 RAM 空间范围。

3.拷贝 stage2 到 RAM 中



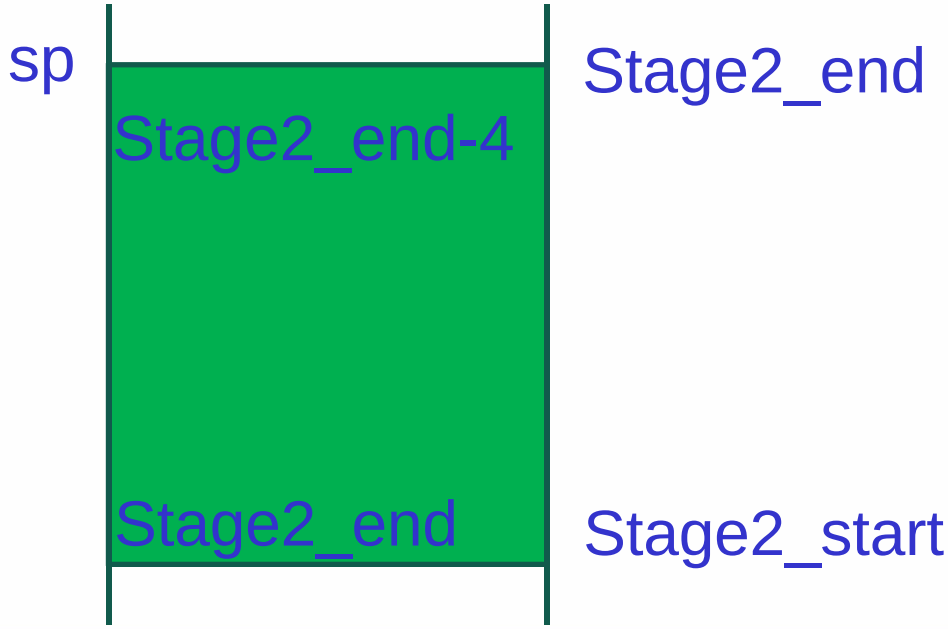
Stage1具体过程

4.设置堆栈指针 sp为执行 C 语言代码作好准备

- sp 指向stage2的RAM空间的最顶端
- 这里的堆栈向下生长， sp指向stage2_end-4
- ARM支持全部4种堆栈方式——满递增、空递增、满递减、空递减

5.跳转到

stage2 的C 入口点



Stage2具体过程

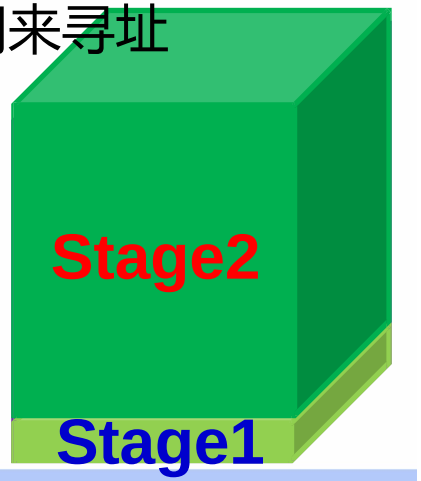
1.初始化本阶段要使用到的硬件设备

- 初始化至少一个串口，以便和终端用户进行I/O 输出信息
- 初始化计时器等
- 设备初始化完成后，可以输出一些打印信息， 程序名字字符串、版本号等。

2.检测系统的内存映射（ memory map ）

是指在整个 4GB 物理地址空间中 有哪些地址范围被分配用来寻址系统的 RAM 单元

- 不同CPU有不同的内存映射，必须准确识别



Stage2具体过程

3.加载内核映像和根文件系统映像

(1) 规划内存占用的布局

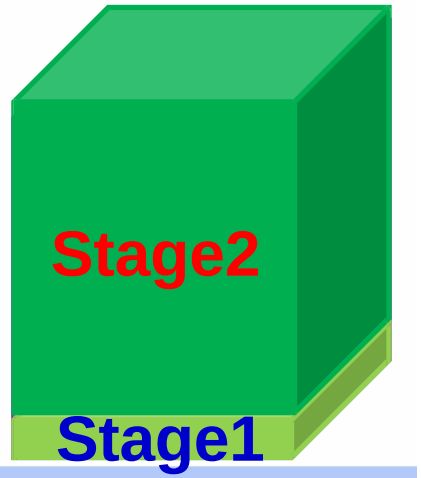
内核映像所占用的内存范围；根文件系统所占用的内存范围

(2)从 Flash 上拷贝内核映像和根文件系统

4.设置内核的启动参数

将内核映像和根文件系统映像拷贝到 RAM 空 间中后，就可以准备启动 Linux 内核了

- 在调用内核之前，需要设置 Linux 内核的启动参数



5.调用内核

- 直接跳转到内核的第一条指令处

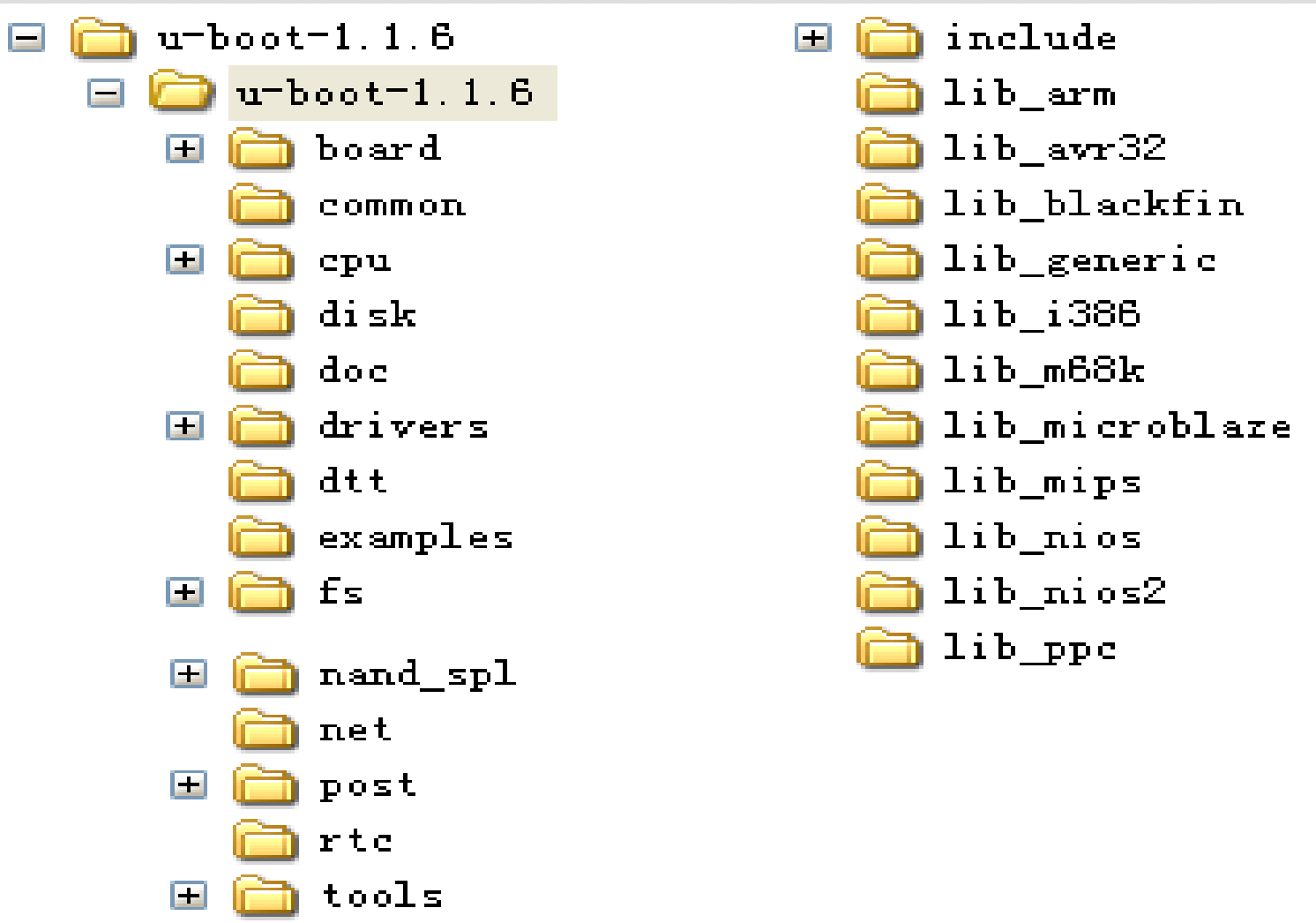
德国DENX软件工程中心U-boot

- ✓ 系统引导：支持NFS挂载、RAMDISK(压缩或 非压缩)形式的根文件系统。
- ✓ 支持目标板环境参数多种存储方式，FLASH、NVRAM、EEPROM
- ✓ 设备驱动:串口、SDRAM、FLASH、以太网、LCD、NVRAM、EEPROM、键盘、USB、PCMCIA、PCI、RTC等驱动支持。
- ✓ 上电自检功能：SDRAM、FLASH大小自动检 测；SDRAM故障检测；CPU型号。

U-Boot顶层目录下有多个子目录，分别存放和管理不同的源程序，这些目录中所存放的文件有其规则，可分为3类

- ✓ 第1类目录与处理器体系结构或者开发板硬件直接相关；
- ✓ 第2类目录是一些通用的函数或者驱动程序；
- ✓ 第3类目录是U-Boot的应用程序、工具或者文 档。

常用Bootloader简介



- 全新的开发板没有任何程序可以执行，无法启动，需要先将U-Boot烧写到Flash中
- 多数嵌入式单板通过处理器的调试接口， 直接对板上的Flash编程

✓ 最简单方式就是通过JTAG电缆，转接到计算机并口连接，把Bootloader下载并烧写到Flash 中

✓ 通常只有全新的开发板需要通过jtag烧写（或者被损坏重新修复后）

✓ 烧写完成后，复位实验板，U-Boot即可启动

常用Bootloader简介

Bootloader除了u-boot，还有redboot，lilo等。Vivi 是韩国mizi公司专门为三星s3c2440芯片设计的Bootloader。

Vivi也可以分为2个阶段，阶段1的代码在arch/s3c2440/head.S中，阶段2的代码从init/main.c的main函数开始。

```
vivi-+-arch-+-s3c2440
|-Documentation
|-drivers-+-serial
|         `-mtd-+-maps
|               |-nor
|               `-nand
|-include-+-platform
|         |-mtd
|         `-proc
|-init
|-lib-+-priv_data
|-scripts-+-lxdialog
|-test
|-util
```

本章主要讲述

7.1 SHELL编程

7.2 BootLoader过程

7.3 各章复习概要

第一章：嵌入式系统的特点，嵌入式系统的组成，嵌入式系统的分类

第二章，计算机系统两种体系结构和两种指令系统的区别；**ARM**内核命名规则的含义，**ARM**微处理器七种运行模式的特点，**ARM**微处理器两种工作状态的区别和切换方法，**CPSR**寄存器控制字分析，**ARM**体系结构中的两种存储格式，**ARM**处理器**MMU**的地址转换过程及虚拟地址到物理地址的转换方法、**ARM**的七种异常类型，异常向量、异常向量地址、异常向量表的含义，**ARM**状态下异常处理过程。

第三章：8种寻址方式的特点，熟悉常见arm汇编指令的格式和功能；能够看懂基本的arm汇编程序，能够结合第五章学习的各种接口，编写简单的接口操作汇编程序。

第四章：熟悉嵌入式存储器的分类，及不同类型存储器的特点和使用场合。弄清嵌入式存储器系统的构成及其存储空间的分布和特点。掌握基本的存储器芯片与嵌入式微处理器芯片的连接方法。

第五章：掌握使用IO端口实现基本数据输入输出的方法，包括相关特殊功能寄存器控制字的分析方法及使用汇编指令实现控制的方法。弄清中断控制器的功能及相关特殊功能寄存器的用途。重点掌握普通外部中断的处理流程及使用汇编程序实现中断初始化和中断处理及中断返回的方法。

围绕S3C2440微处理器芯片，弄清其内部定时器的构成及工作过程。能够根据定时需求，初步设计定时器相关参数并使用汇编指令配置相关特殊功能寄存器。

第六章：交叉编译概念、Linux的子系统结构和功能、常见终端命令

Makefile文件的结构、Linux的调试

第七章：Shell编程基础（含简单示例程序）、Bootloader过程（两阶段）

谢谢各位同学！

Q&A