Linux 操作系统——系统编程

作者：TX

http://tang.chat

2021 年 5 月 10 日

目录

shell 程序设计

- 基本概念
- 常见分类
- 词法定义
- 语法结构
- 脚本文件
- 综合实践

基本概念

Linux 系统 =Linux 内核 (kernel) + 外壳 (shell)

常见分类

最早的 Shell 命令作者：Ken Thompson

- Bourne Shell
  - Bourne Shell (sh)
  - Bourne Again Shell (bash)
  - FreeBSD Shell (sh)
  - Korn Shell (ksh)
  - Debian Almquist Shell (dash)
- C Shell
  - C Shell (csh)，作者：Bill Joy
  - (tcsh)

词法定义

- 空白 (Blank):
  - 空格字符 (Space)
  - 制表字符 (TAB)
- 单词 (Word、Token): 非空白连续字符构成的基本单元，如 echo、-n、"hello world" 等
- 名称 (Name): 以字母或下划线开头、连续字母数字构成的基本单元，如 HOME、PS1 等
- 控制字符 (Control Character): 控制命令执行的字符
  - 分号 (;)
  - 与号 (&)
  - 或号 (|)
  - 小括号 ((、))
  - 换行
- 关键字 (Keyword)、保留字 (Reserved Word): 特殊意义
  - 感叹号 (!)
  - 大括号 ({、})
  - if、fi、else、elif、case、esac、for、do、done、while、until 等

语法结构

- 命令 (Command)
- 列表 (List)
- 分组 (Compound)
- 管道 (Pipeline)
- 重定向 (Redirection)
- 流程控制 (Flow Control)
- 函数 (Function)
- 变量 (Variable)
- 引用 (Quoting)
- 扩展 (Expansion)
- 注释 (Comment)

命令

- 命令类型：
  - 函数命令 (Function)
  - 内建命令 (Built-in Command)
  - 程序命令 (Program)
- 命令结构：
  - 简单命令 (Simple Command)
  - 复杂命令 (Complex Command)：列表、分组、管道、重定向、流程控制、函数定义
- 测试示例：

```
1  echo hello                  # 内建命令 ( echo ) 、简单命令
2  type echo
3  /bin/echo hello             # 程序命令 ( echo ) 、简单命令
4  echo() /bin/echo hello      # 函数命令 ( echo ) 、复杂命令 ( 函数定义 )
5  echo                        # 函数命令 ( echo ) 、简单命令
6  type echo
7  unset -f echo
```

命令——内建命令

- 目录：pwd、cd、chdir、umask、dirs (b)、pushd (b)、popd (b)
- 进程：builtin、command、eval、exec、kill、trap、times、logout (b)、suspend (b)
- 作业：bg、fg、jobs、jobid、wait、disown (bash)
- 函数：.、return、exit、local、getopt、source (b)、caller (b)
- 输出：echo、printf
- 输入：read
- 变量：set、unset、setvar、readonly、export、declare (b)、typeset (b)
- 数组：mapfile (b)、readarray (b)
- 判断：test、false、true
- 循环：shift、break、continue
- 历史：hash、fc、history (b)
- 别名：alias、unalias
- 补全：complete (b)、compgen (b)、compopt (b)
- 设置：ulimit、bind、shopt (b)、enable (b)
- 帮助：type、help (b)

说明：(b) 代表 bash 中特有的命令

命令——内建命令——输入输出

- 输出数据：
  - 语法格式：echo [选项] [参数...]
  - 功能说明：将 0 个或多个参数表示的数据输出至标准输出设备（默认为显示器），默认换行显示
  - 退出状态：成功返回 0，否则非 0
  - 常用选项：
    - -n：取消默认换行
    - -e：启用转义字符
  - 测试示例：

```
1 echo hello world
2 echo -n -e "\x41␣\x10␣b"
```

- 输入数据：
  - 语法格式：read [选项] [变量...]
  - 功能说明：从标准输入设备（默认为键盘）读取数据保存至变量
  - 退出状态：成功返回 0，文件结束返回 1，发生错误返回 2-127，其他情况为信号中断
  - 常用参数：
    - -p：显示提示
    - -t：超时中断
  - 测试示例：

```
1 read -p Name: -t 2 NAME
2 echo $NAME
3 echo $?
```

命令——内建命令——条件测试

- 语法格式：
  - test 表达式
  - [表达式]
- 退出状态：成功返回 0，否则返回 1
- 常用选项：

命令——内建命令——条件测试——测试文件

- 语法格式：

- -e：是否存在指定文件（忽略文件类型）
- -f：是否存在指定普通文件
- -d：是否存在指定目录文件
- -L：是否存在指定符号链接文件
- -b：是否存在指定分块文件
- -c：是否存在指定字符文件
- -k：是否存在指定管道文件
- -S：是否存在指定套接字文件
- -r file：是否存在指定可读文件
- -w file：是否存在指定可写文件
- -x file：是否存在指定可执行文件
- -u file：是否存在指定设置用户标识文件
- -g file：是否存在指定设置分组标识文件
- -k file：是否存在指定设置粘性比特文件
- -O file：是否存在指定所属用户和有效用户标识相同的文件
- -G file：是否存在指定所属群组和有效群组标识相同的文件
- -z file：是否存在指定空文件
- -s file：是否存在指定非空文件
- file1 -ef file2：是否某个文件和另一个文件相同
- file1 -nt file2：是否某个文件比另一个文件较新
- file1 -ot file2：是否某个文件比另一个文件较旧

- 测试示例：

```
1 test -d /bin; echo $?
2 test /sbin/init -ef /lib/systemd/systemd; echo $?
```

命令——内建命令——条件测试——测试字符串

- 语法格式：
  - string：是否存字符串不为空
  - -z string：是否存字符串为空
  - s1 = s2：是否字符串相同
  - s1 != s2：是否字符串不同
- 测试示例：

```
1 S=""; test $S; echo $? test -z $S; echo $?
2 test "S" = ""; echo $?
3 test "S" != ""; echo $?
```

命令——内建命令——条件测试——测试数值

- 语法格式：
  - n1 -eq n2：是否数值相同
  - n1 -ne n2：是否数值不同
  - n1 -gt n2：是否某个数值大于另一个数值
  - n1 -ge n2：是否某个数值大于或等于另一个数值
  - n1 -lt n2：是否某个数值小于另一个数值
  - n1 -le n2：是否某个数值小于或等于另一个数值
- 测试示例：

```
1 test 0 -eq 0; echo $?
2 test 0 -ne 0; echo $?
3 test 1 -gt 0; echo $?
4 test 0 -lt 1; echo $?
```

命令——内建命令——条件测试——测试逻辑

- 语法格式：
  - 表达式 1 -a 表达式 2：是否所有为真
  - 表达式 1 -o 表达式 2：是否某个为真
  - ！表达式 1：是否不为真
- 测试示例：

```
1 test -f /bin -a -f /sbin; echo $?
2 test -z "hello" -o -z ""; echo $?
3 test ! 0 -ne 0; echo $?
```

命令——简单命令

- 语法格式：[变量名称 = 变量值] 命令 [参数 1][, 参数 2]...
- 退出状态：使用“$?”访问退出状态，0 代表成功，1-255 代表失败
- 测试示例：

```
1 man man
2 echo $?
3 man manual
4 echo $?
5 HOME=/ cd
```

命令——复杂命令——列表命令

- 顺序列表命令 (List Command)
  - 语法格式：命令 1 [; | 换行] 命令 2 ...
  - 功能说明：顺序同步执行命令 1、命令 2……
  - 退出状态：最后一个命令的退出状态
  - 测试示例：

```
1 echo hello; echo world; echo $?
2 echo hello; echoo world; echo $?
```

- 后台列表命令 (Background Command)
  - 语法格式：命令 & ...
  - 功能说明：后台异步执行命令
  - 退出状态：0
  - 测试示例：

```
1 vi & vi & vi &; echo $?
2 jobs                    # 查看作业列表
```

- 短路列表命令 (Short-Curcuit Command)
  - 语法格式：命令 1 [&& ||] 命令 2 ...
  - 功能说明：逻辑与 (第 1 条命令成功才执行第 2 条命令)、逻辑或 (第 1 条命令失败才执行第 2 条命令)
  - 退出状态：第一个出错命令或最后一个命令的退出状态
  - 测试示例：

```
1 echoo hello && echo world; echo $?
2 echo hello || echoo world; echo $?
```

命令——复杂命令——分组命令

- 当前 shell 分组命令：
  - 语法格式：{ 命令... ; }
  - 功能说明：在当前 shell 环境中执行大括号内部的命令
  - 退出状态：内部命令的退出状态
  - 注意事项：
    - 大括号属于保留字，必须和其他字符之间用空白分开
    - 内部命令以分号或换行结束
  - 测试示例：

```
1 cd; pwd; { cd /bin; pwd; }; pwd
```

- 子级 shell 分组命令：
  - 语法格式：(命令...)
  - 功能说明：在新的 shell 环境中执行小括号内部的命令
  - 退出状态：内部命令的退出状态
  - 测试示例：

```
1 cd; pwd; (cd /bin; pwd); pwd
```

命令——复杂命令——重定向命令

- 重定向输出
- 重定向输入
- 重定向输入和输出

命令——复杂命令——重定向命令——重定向输出

- 重定向输出
  - 语法格式：命令 [n]> 文件
  - 参数说明：n 为整数表示的文件描述符，其中 0 代表标准输入（输入为键盘）、1 代表标准输出（默认为显示器）、2 代表标准错误（默认为显示器）
  - 测试示例：

```
1 echo hello world > readme.txt
2 cat readme.txt
3 echo hello UNIX 1> readme.txt
4 cat readme.txt
5 ehco hello BSD 2> readme.txt
6 cat readme.txt
```

- 重定向追加输出
  - 语法格式：命令 [n]» 文件
  - 测试示例：

```
1 echo hello linux >> readme.txt
2 echo hello debian >> readme.txt
3 cat readme.txt
```

- 重定向合并输出
  - 语法格式：命令 [n]>&n
  - 测试示例：

```
1 { ehco hello; echo hello; } > readme.txt 2>&1
2 cat readme.txt
```

命令——复杂命令——重定向命令——重定向输入

- 重定向输入
  - 语法格式：命令 [n]< 文件
  - 测试示例：

```
1 cat < readme.tx
```

- 重定向输入本地文档 (Here-Document)
  - 语法格式：

    命令 [n]<< 分隔单词

    ...

    分隔单词
  - 测试示例：

```
1 cat << EOF
2 hello world
3 EOF
```

命令——复杂命令——管道命令

- 语法格式：命令 1 [| 或 |$] 命令 2
- 功能描述：命令 1 的标准输出做为命令 2 的标准输入
- 退出状态：最后一个命令的退出状态
- 测试示例：

```
1 ls -l /etc | less
2 { ehco hello; echo hello; } | wc -l
3 { ehco hello; echo hello; } |& wc -l
```

命令——复杂命令——管道命令——计时

- 语法格式：time [参数] 命令
- 功能描述：计算命令执行的实际时间（real time）、用户时间（user time）、系统时间（system time）
- 退出状态：命令退出状态
- 测试示例：

```
1 time { ehco hello; echo hello; } | wc -l
2 time { ehco hello; echo hello; } |& wc -l
```

命令——复杂命令——流程控制命令

- 判断：
  - 语法格式：
    - if 命令; then 命令; [elif 命令; then 命令;] [else 命令;] fi
    - case 单词 in 模式) 命令;; ... esac
  - 测试示例：

```
1 read X; if [ $X -eq 0 ]; then echo ok; else echo err; fi
2 read X; case $X in [a-z]) echo a;; [0-9]) echo 0;; *) echo ?;; done
```

- 循环：
  - 语法格式：
    - for 变量 [in 单词...;] do 命令; done
    - while 命令; do 命令; done
    - break [次数]
    - continue[次数]
  - 测试示例：

```
1 for i in 1 2 3; do echo hello world; done
2 X=0; while [ $X -lt 6 ]; do echo $X; X=$(($X + 1)); done
3 X=0; while $(true); do echo $X; X=$(($X + 1)); if [ $X -eq 6 ]; then break fi; done
```

命令——复杂命令——函数定义命令

- 语法格式：
  - 函数定义：函数名称 () 命令...
  - 返回状态：return n
- 测试示例：

```
1 welcome() { echo Hello World; return 1; }
2 welcome
3 echo $?
```

参数（Parameter）

- 位置参数：
  - 语法格式：$0，$1，...，$9
  - 测试示例：

```
1 welcome() { echo Hello $1; }
2 welcome tx
```

- 特殊参数：
  - 语法格式：
    - 参数数量：$#
    - 参数列表：$*
    - 参数数组：$@
    - 最近后台命令进程标识：$!
    - 最近命令状态：$?
    - 当前 shell 进程标识：$$
    - 当前 shell 选项设置：$-
  - 测试示例：

```
1 welcome1() { for x in "$*"; do echo $x; done; }
2 welcome2() { for x in "$@"; do echo $x; done; }
3 welcome1 Hello World "Hello World"
4 welcome2 Hello World "Hello World"
```

变量（Variant）

- 查看变量
- 设置变量
- 清除变量
- 系统变量

变量（Variant）——查看变量

- 语法格式：
  - 列出环境变量（全局变量）：env
  - 列出 shell 变量（局部变量）：set
  - 引用指定变量：${ 变量名称 }、$ 变量名称
- 测试示例：

```
1 env
2 set
3 echo $SHELL
```

变量（Variant）——设置变量

- 语法格式：
  - 设置 shell 变量：[set] 变量名称 = 值
  - 设置环境变量：export|$setenv_{csh}$ 变量名称 [= 值]
- 注意事项：等号两端不能有空白字符；若值中有空格，应使用双引号括起来
- 测试示例：

```
1  X=1
2  echo $X
3  X="Hello World"
4  echo $X
5  sh
6  echo $X
7  exit
8  export X
9  sh
10 echo $X
11 exit
```

变量（Variant）——清除变量

- 语法格式：
  - 清除变量：unset 变量名称
  - 清除函数：unset -f 函数名称
- 测试示例：

```
1 X="Hello World"
2 echo $X
3 unset X
4 echo $X
5 ls() { echo ls hacked; }
6 ls
7 unset -f ls
```

变量（Variant）——系统变量

编号	名称	含义	环境变量	shell 变量
1	SHELL	shell 路径	✓	✓
2	TERM	终端名称	✓	✓
3	PWD	工作目录	✓	✓
4	PATH	程序搜索路径	✓	✓
5	USER	用户名称	✓	✓
6	LOGNAME	登录用户名称	✓	✓
7	MAIL	邮件文件名称	✓	✓
8	HOME	用户主目录路径	✓	✓
9	ENV	交互式 shell 初始化文件	✓	✓
10	EDITOR	编辑模式	✓	✓
11	PAGER	分页程序	✓	✓
12	IFS	输入字段分隔字符		✓
13	PS1	主要提示字符串		✓
14	PS2	次要提示字符串		✓
15	PS4	跟踪输出前缀字符串		✓
16	OPTIND	下次选项索引		✓
17	PPID	父进程标识		✓
18	_	上次命令	(f)	✓
19	BLOCKSIZE	分块单位	(f)	(f)

变量（Variant）——系统变量——提示字符串

- 常用转义提示字符：
  - \$：普通用户显示 $ 符号，root 用户显示 # 符号
  - \\：显示反斜杠符号
  - \u：显示用户名称
  - \h：显示主机名称
  - \H：显示完整主机名称（完全限定域名，FQDN，Fully Qualified Domain Main）
  - \w：显示当前工作目录路径
  - \W：显示当前工作目录路径最终目录名称

- 测试示例：

```
1 echo $PS1
2 PS1="\w>"
3 PS1="\u@\h:\w\$"
```

```
1 echo $PS2
2 echo "Hello
3 World"
4 PS2=continue:
5 echo "Hello
6 World"
```

```
1 echo $PS4
2 set -x
3 echo ~
4 PS4=trace:
5 echo ~
6 set +x
7 echo ~
```

选项（Option）

- 语法格式：
  - 查看选项：set (-|+)o
  - 开关选项：set (-|+) 选项
  - 开关选项：set (-|+)o 选项名称参数...
- 常用选项：
  - -o vi：启用 vi 命令行编辑模式选项
  - -o emacs：启用 Emacs 命令行编辑模式选项
  - -o ignoreeof：启用忽略文件结束信号退出 shell 选项
  - -o xtrace | -x：启用调试选项
- 注意事项：-代表开启，+ 代表关闭
- 测试示例：

```
1 set -o
2 set -o vi
3 set -o xtrace
4 echo ~
5 ^D
6 set -o ignoreeof
7 ^D
```

引用 (Quoting)

- 语法格式：
  - 单引号 (Single Quotes)：原样显示每个字符
  - 美元单引号 (Dollar Single Quotes)：保留转义功能，其他字符原样显示
  - 双引号 (Double Quotes)：保留美元符号 ($)、抑音符 (`) 的扩展功能，以及反斜线 (\ ) 后跟美元符号、单引号、双引号、反斜线的转义功能，其他字符原样显示
  - 反斜杠 (Backslash)：转义、续行
- 测试示例：

```
1 echo    'Path:  \n$PWD'
2 echo  $'Path:  \n$PWD'
3 echo  $'P\x61th:  \x0d$PWD'
4 echo    "Path:␣\n$PWD"
5 echo    Path:  \
6 $PWD
```

- 转义字符：
  - 警告 (Alert)：\a
  - 后退 (Backspace)：\b
  - 逃脱 (Escape)：\e
  - 换页 (Formfeed)：\f
  - 新行 (Newline)：\n
  - 回车 (Carriage Return)：\ r
  - 水平制表 (Horizontal Tabulator)：\t
  - 垂直制表 (Vertical Tabulator)：\v
  - 单引号 (Single Quote)：\'
  - 双引号 (Double Quote)：\"
  - 反斜杠：\\
  - 八进制或十六进制数值表示的 ASCII 字符：\ nnn、\xnn
  - 4 位或 8 位十六进制 UNICODE 字符：\xnnnn、\xnnnnnnn

扩展 (Expansion)

- 参数扩展 (Parameter Expansion)
- 命令替换 (Command Substitution)
- 数学扩展 (Arithmetic Expansion)
- 字段分割 (Field Splitting)
- 路径扩展 (Path Expansion)
- 波浪号扩展 (Tilde Expansion，~，波形号、腭化符)

扩展——参数扩展（Parameter Expansion）

- 语法格式：
  - 使用默认值：${参数:-单词}
  - 设置默认值：${参数:= 单词}
  - 指示错误值：${参数:?[单词]}
  - 使用替换值：${参数:+ 单词}
  - 移除最小后缀模式：${参数% 单词}
  - 移除最大后缀模式：${参数%% 单词}
  - 移除最小前缀模式：${参数 # 单词}
  - 移除最大前缀模式：${参数 ## 单词}
  - 字符串长度：${# 参数}
- 测试示例：

```
1 echo ${X1:-default}
2 X1=192.168.56.1; echo ${X1:-default}; echo $X1
3 echo ${X2:=assign}; echo $X2
4 echo ${X3:?error}
5 echo ${X4:+substitute}; echo ${X1:+Substitute}
6 echo ${X1%.*}; echo ${X1%%.*}
7 echo ${X1#*.}; echo ${X1##*.}
```

扩展——命令替换 (Command Substitution)

- 语法格式：
  - 美元小括号：$(命令)
  - 反引号 (Backquote)：‘命令 ‘
- 注意事项：
- 测试示例：

```
1 echo date: $(data)
2 echo data: ‘date‘
```

扩展——数学扩展（Arithmetic Expansion）

- 语法格式：$((表达式))
- 功能说明：
  表达式分类：
  - 常量：十进制、八进制（0 开头）、十六进制（0x 开头）常数
  - 变量
  - 运算符
    - 一元运算符：正号负号（+ -）、逻辑非（!）、按位取反（~）
    - 二元运算符：数学运算符（* / % + -）、按位移动（«» ）、关系运算符（«= »= == ）、按位运算符（&^| ）、逻辑运算符（&&|| ）、赋值运算符（= *= /= += -= «= »= &= ^= |= ）
    - 三元运算符：条件运算符（?: ）
- 测试示例：

```
1 #X=$(head -c 4 /dev/random|hexdump -e '"%u"')
2 echo $(date "+%s")
3 echo $(($(date "+%s")%6))
4 echo $(($(date "+%s")%6+1))
```

扩展——字段分割 (Field Splitting)，空白分割

- 测试示例：

```
1 S="hello:world"
2 echo $S
3 IFS=":"
4 echo $S
5 for i in $S; do echo $i; done;
```

扩展——路径扩展（Path Expansion）

- 语法格式：
  - *：匹配任意数量字符
  - ?：匹配单个字符
  - [...-...]：匹配字符集合
  - [!...-...]：不匹配字符集合
  - [[: 单词:]]：匹配预定义字符集合，man wctype
- 测试示例：

```
1 ls /bin/l*
2 ls /bin/l?
3 ls /bin/[aeiou]*
4 ls /bin/[a-c]*
5 ls /bin/[!a-y]*
6 ls /bin/*[[:digit:]]*
```

扩展——波形号扩展（Tilde Expansion）

- 语法格式：~
- 测试示例：

```
1 echo ~
```

脚本文件

- 文件格式：
  - 文件格式：以“#!”(Sharp Bang) 加 shell 程序路径做为第 1 行、具有执行权限的文本文件
  - 测试示例：

```
1 #!/bin/sh
2 echo hello world
3 echo hello unix
```

- 运行方法：
  - 语法格式：
    - 包含路径的文件名称
    - shell 程序路径文件名称
    - source 文件名称
    - . 文件名称
  - 测试示例：

```
1 ./x.sh
2 /bin/sh x.sh
3 source x.sh
4 . x.sh
```

综合应用

实现简单 HTTP 服务器

1. 实现网络通信
2. 实现静态网站
3. 实现动态网站

综合应用（一）——实现网络通信

- 接收请求报文
- 发送响应报文
- 循环运行服务
- 分离前端后端

综合应用（一）——实现网络通信

- 实现目标：接收请求报文
- 测试实例
  - 运行程序

```
1 nc -l -p 8000
```

  - 浏览网页：

    http://192.168.56.101:8000

综合应用（一）——实现网络通信

- 实现目标：发送响应报文
- 测试实例
  - 编写脚本：tx_srv.v0.1.sh

```
1 #!/bin/sh
2 msg="hello world"
3 echo "HTTP/1.1 200 OK"
4 echo "Content-Length: ${#msg}"
5 echo "Server: TXS/0.1"
6 echo
7 echo -n $msg
```

  - 运行程序

```
1 chmod +x tx_srv.v0.1.sh
2 nc -l -p 8000 -c ./tx_srv.v0.1.sh
```

综合应用（一）——实现网络通信

- 实现目标：循环运行服务
- 测试实例
  - 编写脚本：tx_srv_start.sh

```
1 #!/bin/sh
2 while echo start $1; do
3         if test ! -p pipe; then
4                 mkfifo pipe
5         fi
6         ./tx_srv$1.sh < pipe | nc -l -p 8000 > pipe
7 done
```

  - 运行程序

```
1 chmod +x tx_srv_start.sh
2 ./tx_srv_start.sh .v0.1
```

综合应用（一）——实现网络通信

- 实现目标：分离前端后端
- 测试实例
  - 编写首页：index.htm

```
1  <html>
2  <head>
3  <title>TXServer</title>
4  <meta charset="utf8" />
5  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6  </head>
7  <body>
8  <p>TXServer: Hello, world.</p>
9  </body>
10 </html>
```

  - 修改脚本：tx_srv.v0.1.sh

```
1  #!/bin/sh
2  msg="hello world"
3  msg=$(cat index.htm)
4  echo "HTTP/1.1 200 OK"
5  echo "Content-Length: ${#msg}"
6  echo "Server: TXS/0.1"
7  echo
8  echo -n $msg
```

综合应用（二）——实现静态网站

- 解析 HTTP 请求行
- 解析 HTTP 请求行中的 URI

综合应用（二）——实现静态网站

- 实现目标：解析 HTTP 请求行
- 测试实例
  - 修改脚本：tx_srv.v0.2.sh

```
1  #!/bin/sh
2  read line
3  CR=$(printf "\r")
4  line=${line%$CR}
5  REQUEST_METHOD=${line%% *}
6  tmp=${line#* }
7  REQUEST_URI=${tmp% *}
8  REQUEST_VER=${tmp#* }
9  info="{"
10 info="${info}REQUEST_METHOD:\"$REQUEST_METHOD\","
11 info="${info}REQUEST_URI:\"$REQUEST_URI\","
12 info="${info}SCRIPT_FILENAME:\"$SCRIPT_FILENAME\","
13 info="${info}REQUEST_VER:\"${REQUEST_VER}\","
14 info="${info}}"
15 echo "HTTP/1.1 200 OK"
16 echo "Content-Length: ${#msg}"
17 echo "Server: TXS/0.2"
18 echo "X-Info: $info"
19 echo
20 echo -n $msg
```

综合应用（二）——实现静态网站

- 实现目标：解析 HTTP 请求行中的 URI
- 测试实例
  - 修改脚本：tx_srv.v0.2.sh

```
#!/bin/sh
read line
CR=$(printf "\r")
line=${line%$CR}
REQUEST_METHOD=${line%% *}
tmp=${line#* }
REQUEST_URI=${tmp% *}
REQUEST_VER=${tmp#* }
echo "$REQUEST_URI" | grep "/$" > /dev/null 2>&1
if test $? -eq 0; then
        SCRIPT_FILENAME="$(pwd)${REQUEST_URI}index.html"
else
        SCRIPT_FILENAME="$(pwd)${REQUEST_URI}"
fi
if test -f $SCRIPT_FILENAME; then
        msg="$(cat $SCRIPT_FILENAME)"
else
        msg="<h1>404 Not Found</h1>"
fi
```

```
info="{"
info="${info}REQUEST_METHOD:\"$REQUEST_METHOD\","
info="${info}REQUEST_URI:\"$REQUEST_URI\","
info="${info}SCRIPT_FILENAME:\"$SCRIPT_FILENAME\","
info="${info}REQUEST_VER:\"${REQUEST_VER}\","
info="${info}}"
echo "HTTP/1.1 200 OK"
echo "Content-Length: ${#msg}"
echo "Server: TXS/0.2"
echo "X-Info: $info"
echo
echo -n $msg
```

综合应用（三）——实现动态网站

- 实现用户注册功能前端网页
- 解析 HTTP 的 GET 请求
- 实现用户注册功能后端脚本
- 实现用户登录功能前端网页
- 解析 HTTP 的 POST 请求
- 实现用户登录功能后端脚本

综合应用（三）——实现动态网站

- 实现目标：实现用户注册功能前端网页
- 测试实例
  - 创建网页：tx_acnt_signup.htm

```
1  <html>
2  <head>
3  <title>Account SignUp</title>
4  <meta charset="utf8" />
5  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6  </head>
7  <body>
8  <h2>Account SignUp</h2>
9  <form method="GET" action="tx_acnt_signup.action">
10 <div>Name: </div>
11 <div><input name="acnt_name" value="student"/></div>
12 <div>Pass: </div>
13 <div><input name="acnt_pass" value="alpha=123'" type="password"/></div>
14 <div><input name="acnt_signup" value="SignUp" type="submit"/></div>
15 </form>
16 </body>
17 </html>
```

综合应用（三）——实现动态网站

- 实现目标：解析 HTTP 的 GET 请求
- 测试实例
  - 创建脚本：tx_srv.v0.3.sh

```
#!/bin/sh
info="{"
read line
CR=$(printf "\r")
line=${line%$CR}
REQUEST_METHOD=${line%% *}
tmp=${line#* }
REQUEST_URI=${tmp% *}
REQUEST_VER=${tmp#* }
SCRIPT_NAME=${REQUEST_URI%\?*}
QUERY_STRING=${REQUEST_URI#*\?}
echo "$SCRIPT_NAME" | grep "/$" 2>&1 > /dev/null
if test $? -eq 0; then
        SCRIPT_FILENAME="$(pwd)${SCRIPT_NAME}index.htm"
else
        SCRIPT_FILENAME="$(pwd)${SCRIPT_NAME}"
fi
```

```
if test -f $SCRIPT_FILENAME; then
        echo "$SCRIPT_FILENAME"|grep ".action$" > /dev/null 2>&1
        if test $? -eq 0; then
                export QUERY_STRING
                msg=$(/bin/sh "$SCRIPT_FILENAME")
        else
                msg=$(cat $SCRIPT_FILENAME)
        fi
else
        msg="404 Not Found"
fi
info="${info}REQUEST_METHOD:\"$REQUEST_METHOD\","
info="${info}REQUEST_URI:\"$REQUEST_URI;\","
info="${info}SCRIPT_NAME:\"$SCRIPT_NAME\","
info="${info}QUERY_STRING:\"$QUERY_STRING\","
info="${info}SCRIPT_FILENAME:\"$SCRIPT_FILENAME\","
info="${info}REQUEST_VER:\"${REQUEST_VER}\","
info="${info}}"
echo "HTTP/1.1 200 OK"
echo "Content-Length: ${#msg}"
echo "Server: TXS/0.3"
echo "X-Info: $info"
echo
echo -n $msg
```

综合应用（三）——实现动态网站

- 实现目标：实现用户注册功能后端脚本
- 测试实例
  - 创建脚本：tx_acnt_signup.action

```
#!/bin/bash
msg="<div>QUERY_STRING: $QUERY_STRING</div>"
querys=$(echo "$QUERY_STRING" | tr '&' ' ')
msg="$msg<div>querys: $querys</div>"
for s in $querys; do
        msg="$msg<div>$s</div>"
        key=$(echo "$s" | cut -d '=' -f 1)
        val=$(echo "$s" | cut -d '=' -f 2)
        if test "$key" = "acnt_name"; then
                acnt_name=$val
        elif test "$key" = "acnt_pass"; then
                acnt_pass=$val
        fi
done
msg="$msg<div>acnt_name: $acnt_name</div>"
msg="$msg<div>acnt_pass: $acnt_pass</div>"
```

```
grep "^${acnt_name}:" tx_acnt.dat > /dev/null 2>&1
if test $? -eq 0; then
        msg="$msg<div>Signup Error: Already Existed
else
        echo "$acnt_name:$acnt_pass:$(date +%s)">>t
        msg="$msg<div>Signup OK</div>"
fi
echo $msg
```

综合应用（三）——实现动态网站

- 实现目标：实现用户登录功能前端网页
- 测试实例
  - 创建网页：tx_acnt_signin.htm

```
1  <html>
2  <head>
3  <title>Account SignIn</title>
4  <meta charset="utf8" />
5  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6  </head>
7  <body>
8  <h2>Account SignIn</h2>
9  <form method="POST" action="tx_acnt_signin.action">
10 <div>Name: </div>
11 <div><input name="acnt_name" value="student"/></div>
12 <div>Pass: </div>
13 <div><input name="acnt_pass" value="alpha=123'" type="password"/></div>
14 <div><input name="acnt_signin" value="SignIn" type="submit"/></div>
15 </form>
16 </body>
17 </html>
```

综合应用（三）——实现动态网站

- 实现目标：解析 HTTP 的 POST 请求
- 测试实例
  - 修改脚本：tx_srv.v0.3.sh

```
#!/bin/sh
info="{"
read line
CR=$(printf "\r")
line=${line%$CR}
REQUEST_METHOD=${line%% *}
tmp=${line#* }
REQUEST_URI=${tmp% *}
REQUEST_VER=${tmp#* }
SCRIPT_NAME=${REQUEST_URI%\?*}
QUERY_STRING=${REQUEST_URI#*\?}
while read line; do
        line=${line%$CR}
        if test "$line" = ""; then
                break
        fi
        key=${line%%:*}
        val=${line#*: }
        if test "$key" = "Content-Length"; then
                REQUEST_HEAD_CONTENT_LENGTH=$(($val))
                info="${info}\"REQUEST_HEAD_CONTENT_LENGTH\":${REQUEST_HEAD_CONTENT_LENGTH},"
        else
                info="${info}\"$key\":\"$val\","
        fi
done
if test ! "$REQUEST_HEAD_CONTENT_LENGTH" = ""; then
        if test $REQUEST_HEAD_CONTENT_LENGTH -gt 0; then
                export POST=$(head -c $REQUEST_HEAD_CONTENT_LENGTH)
                info="${info}\"POST\":\"${POST}\","
        fi
fi
```

```
echo "$SCRIPT_NAME" | grep "/$" 2>&1 > /dev/null
if test $? -eq 0; then
        SCRIPT_FILENAME="$(pwd)${SCRIPT_NAME}index.htm"
else
        SCRIPT_FILENAME="$(pwd)${SCRIPT_NAME}"
fi
if test -f $SCRIPT_FILENAME; then
        echo "$SCRIPT_FILENAME"|grep ".action$" > /dev/null 2>&1
        if test $? -eq 0; then
                export QUERY_STRING
                msg=$(/bin/sh "$SCRIPT_FILENAME")
        else
                msg=$(cat $SCRIPT_FILENAME)
        fi
else
        msg="404 Not Found"
fi
info="${info}REQUEST_METHOD:\"$REQUEST_METHOD\","
info="${info}REQUEST_URI:\"$REQUEST_URI;\","
info="${info}SCRIPT_NAME:\"$SCRIPT_NAME\","
info="${info}QUERY_STRING:\"$QUERY_STRING\","
info="${info}SCRIPT_FILENAME:\"$SCRIPT_FILENAME\","
info="${info}REQUEST_VER:\"${REQUEST_VER}\","
info="${info}}"
echo "HTTP/1.1 200 OK"
echo "Content-Length: ${#msg}"
echo "Server: TXS/0.3"
echo "X-Info: $info"
echo
echo -n $msg
```

综合应用（三）——实现动态网站

- 实现目标：实现用户登录功能后端脚本
- 测试实例
  - 创建脚本：tx_acnt_signin.action

```
#!/bin/bash
msg="<div>POST: $POST</div>"
posts=$(echo "$POST" | tr '&' ' ')
msg="$msg<div>posts: $posts</div>"
for s in $posts; do
        msg="$msg<div>$s</div>"
        key=$(echo "$s" | cut -d '=' -f 1)
        val=$(echo "$s" | cut -d '=' -f 2)
        if test "$key" = "acnt_name"; then
                acnt_name=$val
        elif test "$key" = "acnt_pass"; then
                acnt_pass=$val
        fi
done
msg="$msg<div>acnt_name: $acnt_name</div>"
msg="$msg<div>acnt_pass: $acnt_pass</div>"
grep "^$acnt_name:$acnt_pass" tx_acnt.dat > /dev/null 2>&1
if test $? -eq 0; then
        msg="$msg<div>Login OK</div>"
else
        msg="$msg<div>Login Error: Not Existed</div>"
fi
echo $msg
```

系统程序设计

- 开发环境
- 文件管理
- 进程管理
- 综合应用