一、awk条件判断语句

1.条件语句

IF 语句

2.案例：判断语句的练习

if else 语句，用于双重判断

3.for循环有两种格式

二、性能比较

三、知识点补充

sort：排序

sort -r ： 反向排序

sort -n : 以数字的值得大小为排序做依据

uniq ： 去除重复项

uniq -c ： 在每行前加上表示相应行目出现次数的前缀编号

四、shell数组

1.语法格式如下：

2.使用下标来定义数组

3.获取数组中的所有元素

4.获取数组的长度

五、Shell 函数

1.shell中函数的定义格式

2.函数参数-----类似于shell的位置变量

一、域名概述

1、域名解析的作用

2、hosts文件

3、DNS (Domain Name System)域名系统

二、网页基本术语概述

1、网页

2、网站

3、主页

4、域名　　运维工作

5、HTTP　购买证书使用 https

6、URL　-----网址

7、HTML

8、超链接　----大量使用（新网、小说、图片）

9、发布---上线

三、HTML (超文本标签语言)概述

四、网页基本标签

1、头部部分

2、主体部分

(1)标题标签

(2)行控制相关标签

(3)范围标签

(4)图像标签-----需要会用！！！

(5)超链接标签

(6)特殊符号

五、Web 概述

一、awk条件判断语句

1.条件语句

IF 语句

IF 条件语句语法格式如下：

```
if (condition)
    action   #命令
```

也可以使用花括号来执行一组操作：

```
if (condition)
{
    action-1
    action-1
    .
    .
    action-n
}
```

2.案例：判断语句的练习

1.判断第6列是否小于第7列，是就打印$7

```
[root@localhost ~]# awk '{if($6<$7)print$7 "  too high"}' grade.txt
44  too high
26  too high
[root@localhost ~]# awk '{n=$7; if($6<n){print n " too young"}}' grade.txt
44 too young
26 too young
```

2.判断$4是否匹配Brown，是则输出符合其的所有行

```
[root@localhost ~]# awk '{if($4~/Brown/) print $0}' grade.txt
J.Troll   07/99 4842  Brown-3 12 26 26
L.Tansley 05/99 4712  Brown-2 12 30 28
```

3.判断第三列是否匹配48，是则输出整行

```
[root@localhost ~]# awk '{if($3~/48/) print $0}' grade.txt
ley                 05/99 48311 Green   8  40 44
```

J.Lulu 06/99 48317 green 9 24 26
P.Bunny 02/99 48 Yellow 12 35 26
J.Troll 07/99 4842 Brown-3 12 26 26

4.判断第4列是否不匹配Brown，不匹配则输出整行
[root@localhost ~]# awk '{if($4!~/Brown/){print $0}}' grade.txt
ley 05/99 48311 Green 8 40 44
J.Lulu 06/99 48317 green 9 24 26
P.Bunny 02/99 48 Yellow 12 35 26

5.判断$6是否小于$7，是则输出后面的信息
[root@localhost ~]# awk '{if($6<$7){print $1 "\t" "you can do better"}}' grade.txt
ley you can do better
J.Lulu you can do better

6.判断$1是P.Bunny，并且$4是Yellow的行
[root@localhost ~]# awk '{if($1=="P.Bunny" && $4=="Yellow") print $0}' grade.txt
P.Bunny 02/99 48 Yellow 12 35 26

7.判断$1是P.Bunny,或者$4匹配Brown的行
#？？？？？？？？？？“是” 和“匹配”的意思不一样吗?
方式1：
[root@localhost ~]# awk '{if($1=="P.Bunny" || $4~/Brown/) print $0}' grade.txt
P.Bunny 02/99 48 Yellow 12 35 26
J.Troll 07/99 4842 Brown-3 12 26 26
L.Tansley 05/99 4712 Brown-2 12 30 28
方式2：
[root@localhost ~]# awk '{name=$1;belts=$4; if(belts~/Yellow/) print name "is belt" belts}' grade.txt
P.Bunnyis beltYellow
方式3：
[root@localhost ~]# awk '{if($6< 27)print $0}' grade.txt
J.Lulu 06/99 48317 green 9 24 26
J.Troll 07/99 4842 Brown-3 12 26 26

方式4：
[root@localhost ~]# awk '{basefile="27"}{if($6<basefile)print $0}' grade.txt
J.Lulu 06/99 48317 green 9 24 26
J.Troll 07/99 4842 Brown-3 12 26 26

方式5：$6=$6-1 这里只能在第一行-1吗?
[root@localhost ~]# awk '{if($1=="M.Tansley") $6=$6-1;print $1,$6,$7}' grade.txt
M.Tansley 29 28
ley 40 44
J.Lulu 24 26
P.Bunny 35 26
J.Troll 26 26

L.Tansley 30 28

8.打印$6,$7
[root@localhost ~]# awk '{print $6,$7}' grade.txt
30 28
40 44
24 26
35 26
26 26
30 28
[root@localhost ~]# awk '{print $6 $7}' grade.txt
[root@localhost ~]# awk '{print $6--$7}' grade.txt
[root@localhost ~]# awk '{print $6$7}' grade.txt
3028
4044
2426
3526
2626
3028
9.+=的运用
[root@localhost ~]# awk '(tot+=$6);END{print "Club student total points:"tot}'
grade.tx
tM.Tansley 05/99 4712 Brown-2 12 30 28
ley 05/99 48311 Green   8 40 44
J.Lulu   06/99 48317 green   9 24 26
P.Bunny   02/99 48   Yellow 12 35 26
J.Troll   07/99 4842 Brown-3 12 26 26
L.Tansley 05/99 4712 Brown-2 12 30 28
Club student total points:185
[root@localhost ~]# awk '{(tot+=$6)};END{print "Club student total points:"tot}'
grade.txt
Club student total points:185
[root@localhost ~]# awk '{sum+=$6;print sum}' grade.txt
30
70
94
129
155
185
[root@localhost ~]# awk '{sum+=$6}END{print sum}' grade.txt
185
[root@localhost boot]# ls -l | awk '/^[^d]/{print $9"\t"$5}{tot+=$5}END{print "total
KB:"tot}'
#匹配不是d开头的[^d]
或者：[root@localhost boot]# ls -l | awk '/^-/{print $9"\t"$5}{tot+=$5}END{print
"total KB:"tot}'
config-3.10.0-862.el7.x86_64 147819

```
initramfs-0-rescue-a5b881d691c646c087e83e3039a7aef2.img    74082337
initramfs-3.10.0-862.el7.x86_64.img       31715650
symvers-3.10.0-862.el7.x86_64.gz  304926
System.map-3.10.0-862.el7.x86_64          3409143
vmlinuz-0-rescue-a5b881d691c646c087e83e3039a7aef2     6224704
vmlinuz-3.10.0-862.el7.x86_64       6224704
total KB:122109424

[root@localhost boot]# ls -l | awk '/^-/{sum+=$5}END{print sum}'
122109283
[root@localhost boot]# ls -l | awk '/^-/{sum+=$5}END{print "total:"sum " KB"}'
total:122109283 KB

[root@localhost boot]# ls -l | awk '/^-/{sum+=$5}END{print int(sum/1000/1000)}'
122
[root@localhost boot]# ls -l | awk '/^-/{sum+=$5}END{print
"total:"int(sum/1000/1000) " MB"}'
total:122 MB
[root@localhost ~]# df | awk '{sum+=$4}END{print int(sum/1024/1024)"GB"}'
97GB      #int取整
[root@localhost ~]# df | awk 'NR>1{sum+=$4}END{print (sum/1024/1024)"GB"}'
97.4611GB         #去表头

求和
[root@localhost ~]# awk -F: '{sum+=$3}END{print sum}' /etc/passwd
81493
取平均数
[root@localhost ~]# awk -F: '{sum+=$3}END{print sum/NR}' /etc/passwd
1697.77
[root@localhost ~]# df | awk 'NR>1{sum+=$4}END{print int(sum/1024/1024) "GB"}'
97GB
#一般不 加-Th ，后面跟的单位不一样
```

if else 语句，用于双重判断

```
if else 语句，用于双重判断
如果$7大于30，则打印$1 big,否则打印$1 small
[root@localhost ~]# awk '{if($7>30)print $1 "\t" "  big"; else print $1 "\t" "  small"}'
grade.txt
M.Tansley          small
ley     big
J.Lulu          small
P.Bunny     small
J.Troll         small
L.Tansley    small
```

计划任务：

每月第一个周六执行xx.sh任务
crontab -e
00 00 * * 6 [$(cal | awk 'NR==3{print $NF}') -
eq $(date +%d)] && xx.sh
00 00 * * 6 [$(date +%d) -le 7] && xx.sh
第一个周的最后一天不可能大于7，小的话就等到周六就执行

打印第一行
[root@localhost ~]# awk 'NR==1{i=1; while(i<=NF){print NF,$i;i++}}' grade.txt
7 M.Tansley
7 05/99
7 4712
7 Brown-2
7 12
7 30
7 28

用while语句输出1~100并求和
[root@localhost ~]# awk 'BEGIN{i=1; while(i<=100) {print i;i++}}'
[root@localhost ~]# awk 'BEGIN{i=1; while(i<=100) {sum+=i;i++}print sum}'
5050

[root@localhost ~]# df | awk 'NR>1{i=2;while(i<=4){print $i;i++}print "\n"}'
[root@localhost ~]# df | awk 'NR>1{i=2;while(i<=4){printf $i;i++}}'
printf 不换行

[root@localhost ~]# df | awk 'NR>1{i=2;while(i<=4){sum+=$i;i++}print sum}'

#这个数据是偏大的，不够准确
104806400
106802224
108832392
110862560
112892728
121621544
123698216
222199856
222605896
[root@localhost ~]# df | awk 'NR>1{i=2;sum=0;while(i<=4){sum+=$i;i++}print
sum}'
#用sum=0，清零一下，即是所求

练习案例：
[root@localhost ~]# vim file
[root@localhost ~]# cat file

```
aaa   3   5    11   1
bbb    34   5456  767  878
aaa   32   5465  768  232
[root@localhost ~]# awk '$1~/aaa/' file
aaa   3   5    11   1
aaa   32   5465  768  232
[root@localhost ~]# awk '$1~/aaa/{i=2;while(i<=NF){sum+=$i;i++} print sum}' file
20
6517
[root@localhost ~]# awk '$1~/aaa/{i=2;sum=0;while(i<=NF){sum+=$i;i++} print
sum}' file
20
6497
# 注意sum需要清零
[root@localhost ~]# awk 'NR%2==0{i=2;sum=0;while(i<=NF){sum+=$i;i++} print
sum}' file
7135
#偶数行求和
[root@localhost ~]# awk 'NR%2==1{i=2;sum=0;while(i<=NF){sum+=$i;i++} print
sum}' file
20
6497
#奇数行求和
```

3.for循环有两种格式

格式一：

```
for（变量 in 数组）
{语句}
```

```
[root@localhost ~]# awk 'BEGIN{for(i=1;i<=100;i++){sum+=i}{print sum}}'
5050
[root@localhost ~]#  awk 'NR%2==0{for(i=2;i<=NF;i++){sum+=$i}print sum}' file
7135
[root@localhost ~]#  awk '$1~/aaa/{for(i=2;i<=NF;i++){sum+=$i}print sum}' file
20
6517
[root@localhost ~]#  awk '$1~/aaa/{sum=0;for(i=2;i<=NF;i++){sum+=$i}print sum}'
file
20
6497
# 不要忘了sum=0
```

格式二：

```
for （变量；条件；表达式）
{语句}
```


break	当break语句用于while或for语句时，导致退出程序循环
continue	当continue语句用于while或for语句时，使用程序循环移动到下一个迭代
next	能够导致读入下一个输入行，并返回到脚本的顶部。这可以避免对当前输入行执行其他的操作过程
exit	语句使主输入循环退出并将控制转移到END，如果END存在的话。如果没有定义END规则，或在END中应用exit语句，则终止脚本的执行。

二、性能比较

实现相同功能，可以看到awk实现的性能是shell的50倍！

```
[root@localhost ~]# time (awk 'BEGIN{total=0;for(i=0;i<=10000;i++){total+=i;}print total;}')
50005000
real  0m0.003s
user 0m0.000s
sys   0m0.003s
[root@localhost ~]# time(total=0;for i in $(seq 10000);do
total=$(($total+i));done;echo $total;)
50005000

real  0m0.132s
user 0m0.045s
sys   0m0.007s
```

三、知识点补充

查看当前shell最近使用最多的10个命令

sort：排序

sort -r ： 反向排序

sort -n : 以数字的值得大小为排序做依据

uniq ： 去除重复项

uniq -c ： 在每行前加上表示相应行目出现次数的前缀编号

案例1：

```
[root@localhost ~]# awk '{print $6}' grade.txt
30
40
24
35
26
30
[root@localhost ~]# awk '{print $6}' grade.txt | sort
24
```

```
26
30
30
35
40
[root@localhost ~]# awk '{print $6}' grade.txt | sort | sort -n
24
26
30
30
35
40
[root@localhost ~]# awk '{print $6}' grade.txt | sort | sort -n |uniq
24
26
30
35
40
[root@localhost ~]# awk '{print $6}' grade.txt | sort | sort -n |uniq -c
      1 24
      1 26
      2 30
      1 35
      1 40
[root@localhost ~]# awk '{print $6}' grade.txt | sort | sort -rn |uniq -c
      1 40
      1 35
      2 30
      1 26
      1 24
```

案例2：

```
[root@localhost ~]# vim vv.txt
[root@localhost ~]# cat vv.txt
1
2
2
10
10
28
21
21
30
100
[root@localhost ~]# cat vv.txt | sort
1
10
```

```
10
100
2
2
21
21
28
30
[root@localhost ~]# cat vv.txt | sort -n
1
2
2
10
10
21
21
28
30
100
[root@localhost ~]# cat vv.txt | sort -nr
100
30
28
21
21
10
10
2
2
1
[root@localhost ~]# cat vv.txt | sort -nr | uniq
100
30
28
21
10
2
1
[root@localhost ~]# cat vv.txt | sort -nr | uniq -c
      1 100
      1 30
      1 28
      2 21
      2 10
      2 2
      1 1
[root@localhost ~]# cat vv.txt | sort -n | uniq -c
```

```
      1 1
      2 2
      2 10
      2 21
      1 28
      1 30
      1 100
```

案例3：从history抽取使用最多的命令

```
[root@localhost ~]# history | awk '{print $2}' | sort | uniq -c | sort -nr |head
    161 vim
    135 sed
    120 bash
     97 awk
     55 cat
     52 grep
     51 ls
     25 ifconfig
     23 systemctl
     21 cd
```

案例4：

```
[root@localhost ~]# echo "192.168.200.101" | sed 's/\(.*\.\).*/\1/g'
[root@localhost ~]# echo "192.168.200.101" | awk -F. '{print $1"."$2"."$3}'
192.168.200
[root@localhost ~]# echo "192.168.200.101" | sed 's/\(.*\..*\..*\.\).*/\1/g'
# 需要转义的字符前加转义符
[root@localhost ~]# cat access_log | awk '{print $1}'
[root@localhost ~]# awk '{print $1}' access_log
[root@localhost ~]# awk '{print $1}' access_log | sort -n | uniq -c | sort -nr | head
      7 120.4.0.118
      6 106.46.46.82
      5 60.14.51.143
      5 36.99.89.116
      5 27.208.173.122
      5 182.127.111.18
      5 1.57.218.142
      5 1.194.28.157
      5 119.185.61.17
      5 117.136.61.60
```

案例5：linux分析apache日志获取最多访问的前10个IP

apache日志分析可以获得很多有用的信息，现在来试试最基本的，获取最多访问的前10个IP地址及访问次数。

既然是统计，那么awk是必不可少的，好用而高效。

命令如下：

```
awk '{a[$1] += 1;} END {for (i in a) printf("%d %s\n", a[i], i);}' 日志文件 | sort -n | tail
```

首先用awk统计出来一个列表，然后用sort进行排序，最后用tail取最后的10个。

以上参数可以略作修改显示更多的数据，比如将tail加上-n参数等，另外日志格式不同命令也可能需要稍作修改。

当前WEB服务器中联接次数最多的ip地址
#netstat -ntu |awk '{print $5}' |sort | uniq -c| sort -nr

查看日志中访问次数最多的前10个IP
#cat access_log |cut -d ' ' -f 1 |sort |uniq -c | sort -nr | awk '{print $0 }' | head -n 10 |less

查看日志中出现100次以上的IP
#cat access_log |cut -d ' ' -f 1 |sort |uniq -c | awk '{if ($1 > 100) print $0}' | sort -nr |less

查看最近访问量最高的文件
#cat access_log |tail -10000|awk '{print $7}'|sort|uniq -c|sort -nr|less

查看日志中访问超过100次的页面
#cat access_log | cut -d ' ' -f 7 | sort |uniq -c | awk '{if ($1 > 100) print $0}' | less

统计某url，一天的访问次数
#cat access_log|grep '12/Aug/2009'|grep '/images/index/e1.gif'|wc|awk '{print $1}'

前五天的访问次数最多的网页
#cat access_log|awk '{print $7}'|uniq -c |sort -n -r|head -20

从日志里查看该ip在干嘛
#cat access_log | grep 218.66.36.119| awk '{print $1"\t"$7}' | sort | uniq -c | sort -nr | less

列出传输时间超过 30 秒的文件
#cat access_log|awk '($NF > 30){print $7}' |sort -n|uniq -c|sort -nr|head -20

列出最最耗时的页面(超过60秒的)
#cat access_log |awk '($NF > 60 && $7~/\.php/){print $7}' |sort -n|uniq -c|sort -nr|head -100

四、shell数组

Shell 数组用括号来表示，元素用"空格"符号分割开

1.语法格式如下：

```
array_name=(value1 ... valuen)
```

2.使用下标来定义数组

```
array_name[0]=value0
array_name[1]=value1
```

```
array_name[2]=value2
```

3.获取数组中的所有元素

使用@ 或 * 可以获取数组中的所有元素

4.获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同

案例1：

```
[root@localhost ~]# name=(zhangsan lisi wangwu zhaoliu sunqi)
[root@localhost ~]# add[0]=beijing
[root@localhost ~]# add[1]=hsanghai
[root@localhost ~]# add[2]=guangzhou
[root@localhost ~]# add[3]=shanzhen
[root@localhost ~]# echo ${name[@]}
zhangsan lisi wangwu zhaoliu sunqi
[root@localhost ~]# echo ${name[*]}
zhangsan lisi wangwu zhaoliu sunqi
[root@localhost ~]# echo ${add[@]}
beijing hsanghai guangzhou shanzhen
[root@localhost ~]# echo ${add[0]}
beijing
```

案例2：

```
[root@localhost ~]# vim jj.sh
1 #auther:liruifang
2 my_array=(A B C D)
4 echo "第一个元素为: ${my_array[0]}"
5 echo "第二个元素为: ${my_array[1]}"
6 echo "第三个元素为: ${my_array[2]}"
7 echo "第四个元素为: ${my_array[3]}"
[root@localhost ~]# ./jj.sh
第一个元素为: A
第二个元素为: B
第三个元素为: C
第四个元素为: D
[root@localhost ~]# vim jj.sh
  1 #auther:liruifang
  2 my_array=(A B C D)
  4 echo "第一个元素为: ${my_array[0]}"
  5 echo "第二个元素为: ${my_array[1]}"
  6 echo "第三个元素为: ${my_array[2]}"
  7 echo "第四个元素为: ${my_array[3]}"
  8 echo "数组的元素为: ${my_array[*]}"
[root@localhost ~]# ./jj.sh
第一个元素为: A
第二个元素为: B
```

```
第三个元素为: C
第四个元素为: D
数组的元素为: A B C D
[root@localhost ~]# echo ${#add[@]}
4
[root@localhost ~]# ./jj.sh
第一个元素为: A
第二个元素为: B
第三个元素为: C
第四个元素为: D
数组的元素为: A B C D
数组元素个数为: 4
```

案例3：

```
[root@localhost ~]# vim name.txt       # 这里面写入所有人的名字，如果里面某个人的名字出现的次数越多，那么他回答问题的几率越大
[root@localhost ~]# vim nn.sh
 1 #!/bin/bash
 2 i=0
 3 for n in $(cat name.txt)
 4 do
 5     names[$i]=$n
 6     let i++
 7 done
 9 num=$(echo $RANDOM % ${#names[@]})
10 echo "${names[$num]} 请回答问题"
```

五、Shell 函数

linux shell 可以用户定义函数，然后在shell脚本中可以随便调用

1.shell中函数的定义格式

```
[ function ] funname [()]   # 小括号一般不忽略；前面的function可以不写
{
    action;    # 函数中的动作；具体命令
    [return int;]   # 返回值（整数:0~255），即函数的状态，是成功或失败
}
```

注意：函数的名字可以随便起，但不能和命令冲突，比如：命令是cat，函数名不能是cat

案例：作业练习题

```
[root@CentOS6-node1 ~]# cat case-ip-set.sh
#!/bin/bash
#定义变量
ipfile="/etc/sysconfig/network-scripts/ifcfg-eth0"
read -p "请输入IP地址: " ip
read -p "请输入子网掩码: " mask
read -p "请输入网关: " gw
status=`cat $ipfile | grep BOOTPROTO | awk -F "=" '{print $2}'`
```

```
#定义函数
dhcp(){
cat << END >> $ipfile
IPADDR=$ip
NETMASK=$mask
GATEWAY=$gw
END
}

static(){
sed -i "s/IPADDR=.*/IPADDR=$ip/g" $ipfile
sed -i "s/NETMASK=.*/NETMASK=$mask/g" $ipfile
sed -i "s/GATEWAY=.*/GATEWAY=$gw/g" $ipfile
}

#判断状态
case "$status" in
dhcp)
    sed -i '/^BOOT/ s/BOOTPROTO=dhcp/BOOTPROTO=static/g' $ipfile
dhcp
service network restart
exit 0
;;
none|static)
static
service network restart
exit 0
;;
*)
echo "配置有误，请检查配置文件: $ipfile"
exit 1
esac
```

案例：

```
[root@localhost ~]# vim lol.sh
  1 nini () {
  2 echo "mmm"
  3 echo "nnn"
  4 echo "kkk"
  5 }
  6 echo "碰到了ii:"
  7 nini
  8 echo "碰到了pp:"
 9 nini
[root@localhost ~]# bash lol.sh
碰到了ii:
```

```
mmm
nnn
kkk
碰到了pp:
mmm
nnn
kkk
```

升级版：脚本

```
[root@localhost ~]# vim lol.sh
  1 #!/bin/bash
  3 nini () {
  4 echo "mmm"
  5 echo "nnn"
  6 echo "kkk"
  7 return 2     # 定义返回值，默认0表示最后一条命令执行成功，也可定义其他数
  8 }
  9 echo "碰到了ii:"
 10 nini
 11 echo $?
[root@localhost ~]# bash lol.sh
碰到了ii:
mmm
nnn
kkk
2
```

小计算器---案例

```
[root@localhost ~]# vim lol.sh
  1 #!/bin/bash
  2
  3 tot () {
  4         read -p "输入第一个数字:" a        #这里的数字不能超过255，因为return后面有限
制
  5         read -p "输入第二个数字:" b
  6         sum=$[$a+$b]
  7         return $sum
  8 }
  9
 10 tot
 11 echo "结果为：$?"       # 返回值需与echo配合使用
[root@localhost ~]# bash lol.sh
输入第一个数字:12
输入第二个数字:45
结果为：57
```

2.函数参数-----类似于shell的位置变量

- 函数传参
- 函数在shell中，更多的是把命令按照功能做区分
- 当有多条命令来完成一个功能时，来定义函数

```
[root@localhost ~]# vim lol.sh
#!/bin/bash
tot () {
        sum=$[$1+$2]
        return $sum
}
tot 12 34
echo "结果为：$?"
```

Web网络服务

一、域名概述

1、域名解析的作用

方便记忆，IP 地址不容易记忆，但是域名更加直观。

2、hosts文件

早起使用hosts文件进行域名的解析

Linux中hosts文件存放路径为/etc/hosts

Windows系统中存放路径为C:\Windows\System32\drivers\etc\hosts内

但后期，随着Internet网上的网站发展迅速，一个小小的hosts文件以不足以存放

再加上主机名称重复、主机维护困难等问题，出现了DNS域名解析服务

3、DNS (Domain Name System)域名系统

(1)两大特点:分布式、层次性

(2)域名空间结构:根域、顶级域(国家/地区域名)、二级域

(3)完整域名格式: FQDN= 主机名.DNS后缀，例: www.baidu.com.

二、网页基本术语概述

1、网页

纯文本格式文件，其编写语言为HTML，在用户的浏览器中被“翻译”成网页形式显示出来

2、网站

网站是由一个一个页面构成的，是多个网页的结合体

3、主页

打开网站后出现的第一个网页称为网站主页(或首页 index.html)

4、域名　　运维工作

浏览网页时输入的网址(例如:www.sina.com.cn/)

5、HTTP　购买证书使用 https

用来传输网页的通信协议(超文本传输服务)

6、URL -----网址

是一种万维网寻址系统（统一资源定位符）

http://www.baidu.com./jpg/1.jpg　　# 协议 域名 目录 文件

报错提交给开发人员

URI与URL之间的区别与联系

URI就是一种资源定位机制，它是比较笼统地定位了资源，并不局限于客户端和服务器，而URL就定位了网上的一切资源，只要是网上的资源，都有唯一的URL

总结如下：

1.简写：

URI (uniform resource identifier)统一资源标志符；

URL(uniform resource location)统一资源定位符（或统一资源定位器）；

URN(uniform resource name)统一资源命名。

2.URI 和 URL 的比较

a.

URI是一个相对来说更广泛的概念，URL是URI的一种，是URI命名机制的一个子集，可以说URI是抽象的，而具体要使用URL来定位资源。

b.

Web上的每一种资源如：图片、文档、视频等，都是由URI定位的，这里所谓的定位指的是web上的资源相对于主机服务器来说，存放在服务器上的具体路径。

c.

URL是internet上用来描述信息资源文件的字符串，用在客户程序和服务器上，定位客户端连接服务器所需要的信息，它不仅定位了这个信息资源，而且定义了如何找到这个资源。

3.我的白话理解

URI就是一种资源定位机制，它是比较笼统地定位了资源，并不局限于客户端和服务器，而URL就定位了网上的一切资源，只要是网上的资源，都有唯一的URL.

7、HTML

用来编写网页的超文本标记语言

8、超链接 ----大量使用（新网、小说、图片）

将网站中不同网页链接起来的功能

9、发布---上线

将制作好的网页上传到服务器供用户访问的过程

备份 /var/www/html/*

当网站出现错误的时候，首先将旧版本换上，将新版本的问题提交给开发人员慢慢解决

三、HTML (超文本标签语言)概述

1、HTML

Hyper Text Markup Language

编写网页的语言

网页的“源码

ps：H5（前端--网页）

2、浏览器

“解释和执行”

HTML源码的工具

3、HTML文档的结构

头部部分

标题部分

主体部分

网页内容：包括文本、图像等

四、

1、头部部分

网页摘要信息的作用:

有利于浏览器解析及搜索引擎的搜索

<title> 标题标签

<meta>标签 (meta-information) 针对搜索引擎和更新频度的描述和关键词

ps:技术性帖子之所以排的靠前是因为它的关键词定义的很准确,关键词也得买

看钱投入的多少

示例:

```
<head>
<title>我的测试网页</title>
<meta name="keywords" content="云计算,Linux,网络服务,T技术"/>
</head>
```

2、主体部分

(1)标题标签

<h1>-<h6> <h1>字号最大

<h6>字 号最小

示例:

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
</body>
```

(2)行控制相关标签

<p>段落标签

示例:

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
<p>这是一一个段落</p>
</body>
```


换行标签

示例:

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
<p>这是一一个段落</p>
测试换行标签<br/>
</body>
```

(3)范围标签

范围标签，组合文档中的行内元素

示例:

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
```

这是一一个段落
测试换行标签
范围测试
范围测试
不加入其他属性的设置，无变化
</body>

(4)图像标签-----需要会用！！！

<img>图像标签

示例：

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
<p>这是一个段落</p>
测试换行标签<br/>
<span>范围测试
范围测试</span>
<img src="E:\test.jpg"/>
</body>
```

(5)超链接标签

<a>超链接标签

示例：

```
<body>
<h1>一级 标题</h1>
<h2>二级标题</h2>
<p>这是一个段落</p>
测试换行标签<br/>
<span>范围测试
范围测试</span>
<img src="E:\test.jpg"/>
<a href="http://www.benet.wang" >幽幽林</a>
</body>
```

(6)特殊符号

	空格符号
&quot	引号
&copy	公司版权
&gt	>大于号

示例：

"sofia.com" bbb
 ©Amber.Inc版权所有

案例：

```
<html>
<head>
    <title> 这是李瑞芳的第一个网页 </title>
```

```
        <meta name="keywords" content=
        "云计算，linux，网络服务，IT技术"/>
</head>
<body>
<h1>网站开发</h1>
这是我的一个网页,编写网页很<span style="color:red;font-size:30px">快乐</span>~
<br />
<img src="111.jpg"/>
<a href="https://www.bilibili.com/video/BV1gs41137kb"target="_black"><br />
运动多久能瘦？瘦多少斤？坚持不住怎么办？没时间怎么办？</a>
</body>
</html>
```

五、Web 概述

Web内容储存在Web服务器（维护比较多的一种）上

最简单的Web资源就是Web服务器文件系统中的**静态文件**

```
# 静态的含义是文件本身不会动，不是图片文字不会动的意思
```

静态文件可以包含任意内容 ：

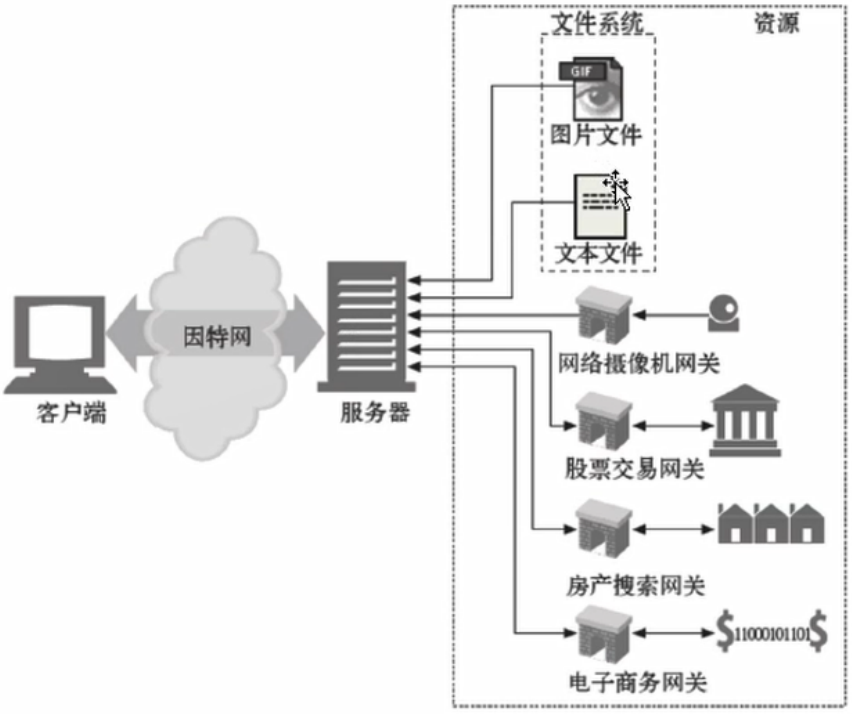
文本文件、HTML文件、

微软的Word文件、Adobe的Acrobat文件、

JPEG 图片文件、AVI电影文件

资源不一定是静态文件，资源还可以是根据需要生成内容的**软件程序**。

这些动态内容资源可以根据你的身份、所请求的信息或每天的不同时段来产生内容。



1、Web1.0与Web2.0

Web1.0------公司产品介绍

以编辑为特征，网站提供给用户的内容是编辑处理后提供的

然后用户阅读网站提供的内容

这个过程是网站到用户的单向行为

Web2.0------主流

更注重用户的交互作用，

用户既是网站内容的消费者(浏览者)，也是网站内容的制造者

Web2.0 加强了网站与用户之间的互动，网站内容基于用户提供

网站的诸多功能也由用户参与建设，实现了网站与用户双向的交流与参与

2、静态网页与动态网页

(1)静态网页

- 纯粹HTML格式的网页--------“静态网页”
- 静态网页：标准的HTML文件
- **扩展名：.htm、.html,**
- 可包含文本、图像、声音、FLASH 动画、客户端脚本和Activex控件及JAVA小程序等
- 静态网页是网站建设的基础，早期的网站一般都是静态网页制作的
- 静态网页相对于动态网页而言，是没有后台数据库、不含程序和不可交互的网页
- 静态网页相对更新起来比较麻烦，适用于一般更新叫少的**展示型网站**（公司首页）

1.特点：

每一个静态网页都有一个固定的URL

URL以.htm、.html、.shtmt等常见形式为后缀，而且不含有“?”

网页内容一经发布到网站服务器上，无论是否有用户访问

每个静态网页都是存在在网站服务器上的

静态网页是实实在在保存在服务器上的文件

每个网页都是一个独立的文件

2.优点：

- 静态网页的内容相对稳定，因此容易被搜索引擎检索
- 页面浏览速度迅速，过程无需连接数据库，开启页面速度快于动态页面

3.缺点：

- 静态网页没有数据库的支持，在网站制作和维护方面工作量较大
- 因此当网站信息量很大时完全依靠静态网页制作方式比较困难
- 静态网页的交互性较差，在功能方面有较大的限制

(2)动态网页----技术不同

- 网页URL的后缀是以**.aspx、 .asp、 .jsp、 .php、 .perl、 .cgi** 等形式为后缀
- 动态网页网址中有一个标志性的符号——“?”
- 动态网页显示的内容随时间、环境或者数据库操作的结果而发生改变
- 动态网页与网页上的各种动画、滚动字幕视觉上的动态效果没有直接关系
- 只要是采用了动态网站技术生产的网页都可以称为“**动态网页**”。

- 是基本的html语法与Java、PHP等高级程序设计语言、数据库编程等多种技术的融合
- 动态网页以**数据库技术**为基础
- 动态网页不是独立存在于服务器上的网页文件，只有当用户请求时，服务器才先查找，然后返回给用户一个完整的网页

1.作用：

- 实现对网站内容和风格的高效、动态和交互式管理
- 大大降低网站维护的工作量
- 实现更多的功能： 用户注册、用户登录、在线调查、用户管理、订单管理等

2.动态页面中的“?” --- 分隔符

对搜索引擎检索存在一定的问题：

- 搜索引擎一般不可能从一个网站的数据库中访问全部网页
- 处于技术方面的考虑，搜索之中不去抓取网址中“?”后面的内容

采用动态网页的网站，在进行搜索引擎推广时，做一定的技术处理(伪静态)，以适应搜索引擎的要求。

另附件：

判断数字是奇数还是偶数：

```
$ awk 'BEGIN {num = 10; if (num % 2 == 0) printf "%d 是偶数\n", num }'
```

输出结果为：

```
10 是偶数
```

IF - ELSE 语句

IF - ELSE 条件语句语法格式如下：

```
if (condition)
    action-1
else
    action-2
```

在条件语句 condition 为 true 时只需 action-1，否则执行 action-2。

```
$ awk 'BEGIN {
   num = 11;
   if (num % 2 == 0) printf "%d 是偶数\n", num;
   else printf "%d 是奇数\n", num
}'
```

输出结果为：

```
11 是奇数
```

IF - ELSE - IF

我们可以创建多个 IF - ELSE 格式的判断语句来实现多个条件的判断：

```
$ awk 'BEGIN {
a=30;
```

```
if (a==10)
  print "a = 10";
else if (a == 20)
  print "a = 20";
else if (a == 30)
  print "a = 30";
}'
```

输出结果为:

```
a = 30
```

循环

For

For 循环的语法如下：

```
for (initialisation; condition; increment/decrement)
    action
```

for 语句首先执行初始化动作(initialisation)，

然后再检查条件(condition)。

如果条件为真，则执行动作(action)，

然后执行递增(increment)或者递减(decrement)操作。

只要条件为 true 循环就会一直执行。

每次循环结束都会进条件检查，若条件为 false 则结束循环。

使用 For 循环输出数字 1 至 5:

```
$ awk 'BEGIN { for (i = 1; i <= 5; ++i) print i }'
```

输出结果为:

```
1
2
3
4
5
```

While

While 循环的语法如下：

```
while (condition)
    action
```

While 循环首先检查条件 condition 是否为 true ，

若条件为 true 则执行动作 action。

此过程一直重复直到条件 condition 为 flase 才停止。

While 循环输出数字 1 到 5 的例子：

```
$ awk 'BEGIN {i = 1; while (i < 6) { print i; ++i } }'
```
输出结果为:
```
1
2
3
4
5
```
Break

break 用以结束循环:

当计算的和大于 50 的时候使用 break 结束循环:
```
$ awk 'BEGIN {
  sum = 0; for (i = 0; i < 20; ++i) {
    sum += i; if (sum > 50) break; else print "Sum =", sum
  }
}'
```
输出结果为:
```
Sum = 0
Sum = 1
Sum = 3
Sum = 6
Sum = 10
Sum = 15
Sum = 21
Sum = 28
Sum = 36
Sum = 45
```
Continue

Continue 语句用于在循环体内部结束本次循环，从而直接进入下一次循环迭代。

输出 1 到 20 之间的偶数:
```
$ awk 'BEGIN {for (i = 1; i <= 20; ++i) {if (i % 2 == 0) print i ; else continue} }'
```
输出结果为:
```
2
4
6
8
10
12
14
16
18
20
```
Exit

Exit 用于结束脚本程序的执行。

该函数接受一个整数作为参数表示 AWK 进程结束状态。 如果没有提供该参数，其默认状态为 0。

下面例子中当和大于 50 时结束 AWK 程序。

```
$ awk 'BEGIN {
  sum = 0; for (i = 0; i < 20; ++i) {
    sum += i; if (sum > 50) exit(10); else print "Sum =", sum
  }
}'
```

输出结果为：

```
Sum = 0
Sum = 1
Sum = 3
Sum = 6
Sum = 10
Sum = 15
Sum = 21
Sum = 28
Sum = 36
Sum = 45
```

让我们检查一下脚本执行后的返回状态：

```
$ echo $?
```

执行上面的命令可以得到如下的结果：

```
19
```

输出1~100质数

```
[root@localhost ~]# cat zhishu.sh
#!/bin/bash
for i in {1..1000}
do
n=$(factor $i | awk '{print NF}')
[ $n -le 2 ] && echo "质数: $i"
done
```

生成随机密码

```
[root@localhost ~]# date +%N%N | head -c 10
8248403918
[root@localhost ~]# mkpasswd -l 10 -d 0 -s 0 -c 1 -C 1
ubtbinrXpm
[root@localhost ~]# date +%s%N | md5sum | head -c 10
024156f5b3
[root@localhost ~]# mkpasswd -l 10
YU5am=x6zr
```

10、请写出下列shell脚本：使用for循环在/opt下通过随机小写10位长度字母加上(+)固定字符串test批量创建10个html文件，并且html大写，创建完成后将test全部改为test_done（for循环实现）：

```
[root@localhost ~]# cat a.sh
#!/bin/bash
for i in {1..10}
do
str=$(date +%N%s | tr '[0-9]' '[a-z]' | head -c 10)
cd /opt
touch $str-test.HTML
rename test test_done $str-test.HTML
done
```

11、随机生成10位数字，随机生成10位字母，随机生成10位字母+数字的混合, 随机生成10位字母+数字的混合+特殊符号。

```
[root@localhost ~]# date +%N%N | head -c 10
8248403918

[root@localhost ~]# mkpasswd -l 10 -d 0 -s 0 -c 1 -C 1
ubtbinrXpm

[root@localhost ~]# date +%s%N | md5sum | head -c 10
024156f5b3

[root@localhost ~]# mkpasswd -l 10
YU5am=x6zr
```

部署Nginx web软件（最新版本） 脚本

```
wget  http://nginx.org/download/nginx-1.17.9.tar.gz
```