10 循环控制语句

循环控制语句

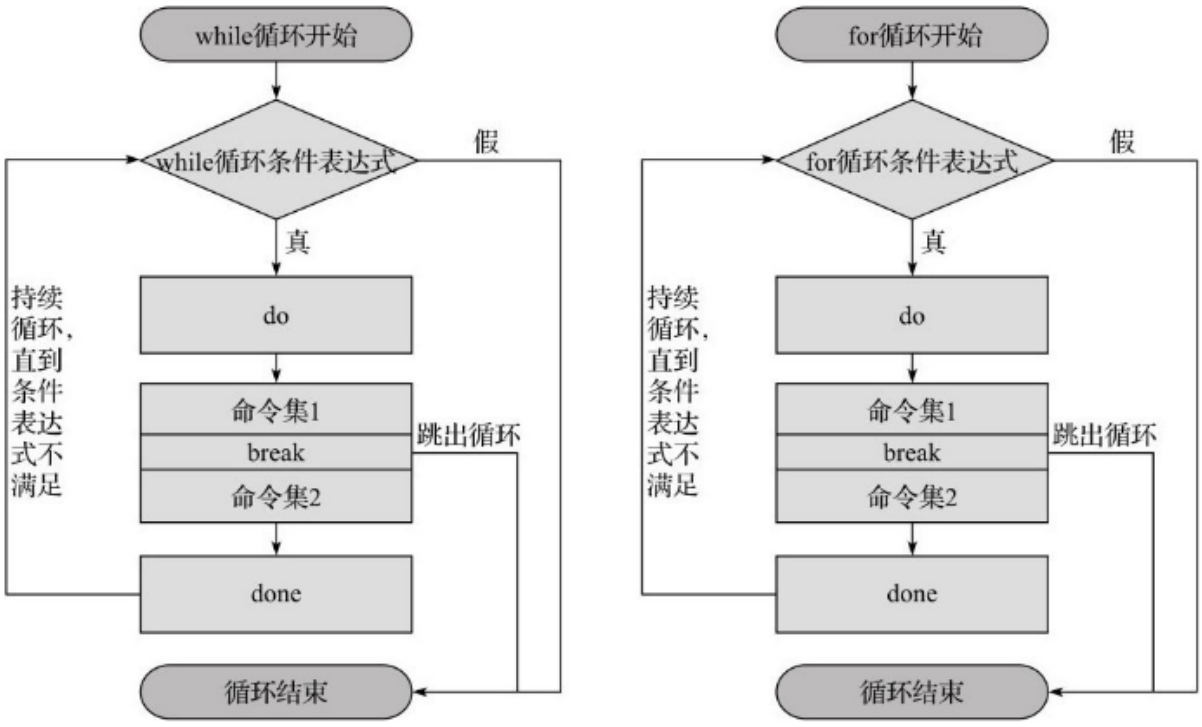
一、循环中断与退出机制

在Shell脚本中，有时需要立即从循环中退出，如果是**退出循环**，则可使用 `break` 循环控制语句；如果是**退出本次循环执行后继循环**，则可使用 `continue` 循环控制语句；如果是**终止所有语句并退出当前脚本**，则可使用 `exit` 语句。

1.1 break语句

`break` 语句可以应用在 **`for`** 、 **`while`** **和** **`until`** 循环语句中，用于**强行退出循环**，也就是**忽略循环体**中任何其他**语句**和**循环条件**的限制。

`while` 循环和 `for` 循环中 `break` 的功能执行流程逻辑如下图所示。



案例：break语句

循环输出1–5。

```shell
[root@shell ~]# vi no_break.sh
#!/bin/bash
for loop in {1..5}
do
    echo "The value is ${loop}"
done
[root@shell ~]# . no_break.sh
The value is 1
The value is 2
The value is 3
The value is 4
The value is 5
```

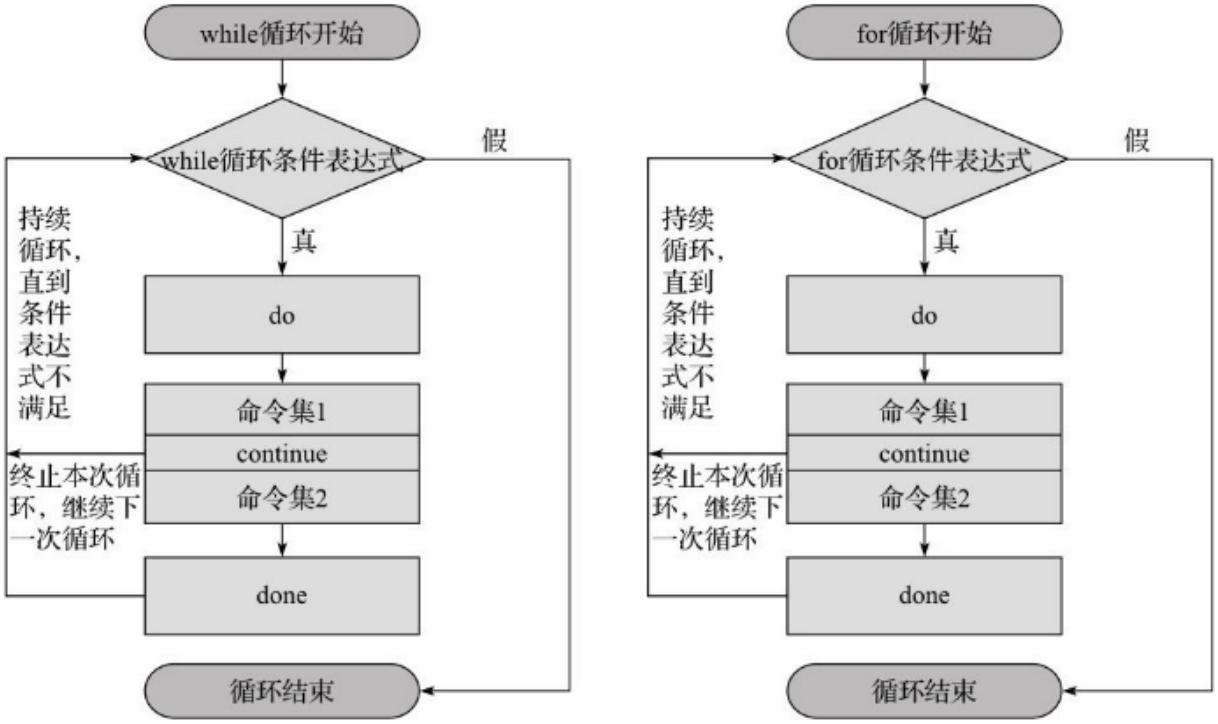
改进程序，遇到偶数就停止输出。

```shell
[root@shell ~]# vi break.sh
#!/bin/bash
for loop in {1..5}
do
    if [[ $(($loop % 2)) -eq 0 ]] ;then
        break
    else
        echo "The value is ${loop}"
    fi
done
[root@shell ~]# . break.sh
The value is 1
```

1.2 `continue` 语句

`continue` 循环控制语句应用在 `for` 、 `while` 和 `until` 语句中，用于让脚本**结束单次循环**（**跳过**其后面的语句），**执行下一次循环**。

`while` 循环和 `for` 循环中 `continue` 的功能执行流程逻辑如下图所示。



案例：continue语句

输出奇数。

Shell | 复制代码

```shell
[root@shell ~]# vi continue.sh
#!/bin/bash
for loop in {1..5}
do
    if [[ $(($loop % 2)) -eq 0 ]] ;then
        continue
    else
        echo "The value is ${loop}"
    fi
done
[root@shell ~]# . continue.sh
The value is 1
The value is 3
The value is 5
```

1.3 `exit` 语句

`exit` 语句用于**终止所有语句并退出当前脚本**。除此之外，`exit` 还可以返回上一次程序或命令的**退出状态码**给当前Shell。

Shell中运行的每个命令都使用**退出状态码**（exit status）来告诉shell它完成了处理。退出状态码是一个 `0~255` 之间的**整数**值，在命令结束时由命令传回shell。

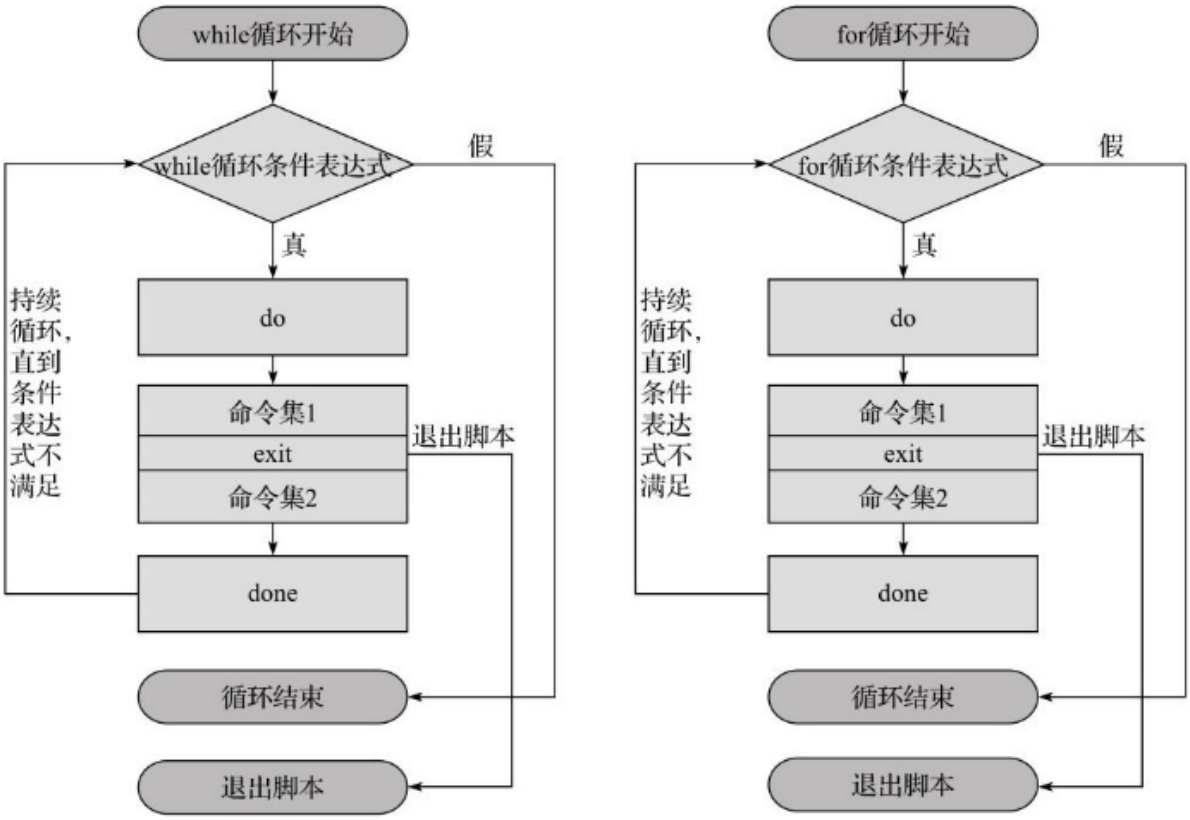
退出状态码的**最大只能是255**。

若其大于255，则返回该值除以256后得到的余数（取模）。

典型的退出状态码及其意义如下表所示。

状态码	描述
0	命令成功结束
1	通用未知错误
2	误用Shell命令
126	命令不可执行
127	没找到命令
128	无效退出参数
128+x	Linux信号x的严重错误
130	命令通过Ctrl+C控制码越界
255	退出码越界

`while` 循环和 `for` 循环中 `exit` 的功能执行流程逻辑如下图所示。



案例：exit语句

遇到偶数就退出脚本。

Shell | 复制代码

```
[root@shell ~]# vi exit.sh
#!/bin/bash
for loop in {1..5}
do
    if [[ $(($loop % 2)) -eq 0 ]] ;then
        exit
    else
        echo "The value is ${loop}"
    fi
done
#注意这里需要用bash起子进程运行脚本，否则会退出当前shell
[root@shell ~]# bash exit.sh
The value is 1
```

⚠️ 为了方便验证结果，需要用**bash命令**起子进程**运行脚本**或者赋予**脚本运行权限**，否则会退出当前shell

利用 `exit` 语句设置退出状态码。

Shell | 复制代码

```
[root@shell ~]# vi exit5.sh
#!/bin/bash
for loop in {1..5}
do
    if [[ $(($loop % 2)) -eq 0 ]] ;then
        exit 5
    else
        echo "The value is ${loop}"
    fi
done
[root@shell ~]# bash exit5.sh
The value is 1
[root@shell ~]# echo $?
5
```

验证大于255的状态。

Shell | 复制代码

```
[root@shell ~]# vi exit260.sh
#!/bin/bash
for loop in {1..5}
do
    if [[ $(($loop % 2)) -eq 0 ]] ;then
        exit 260
    else
        echo "The value is ${loop}"
    fi
done
[root@shell ~]# bash exit260.sh
The value is 1
[root@shell ~]# echo $?
4
```

二、利用循环实现并发控制

默认情况下，Shell命令是**串行方式自上而下**执行的，但如果有大批的命令需要执行，串行就会浪费大量的时间，这时就需要Shell并发执行了。

Shell并发控制有多种方法，这里介绍两种方法，分别为**for循环**实现和**for循环后台**执行，这两种方法在生产场景中会经常使用，都是常见的技术要点。

2.1 for循环并发控制基本语法

for循环实现Shell的并发控制基本语法为:

```shell
=========for 循环============
for  条件测试
do
  循环体
done
=========当条件为真，执行循环体========
```

for后台循环实现Shell的并发控制基本语法为:

```shell
=========for循环========
for 条件测试
do
{
    循环体
  }&
done
===========当条件为真时，执行循环体，&表示后台执行==============
```

2.2 for循环实现Shell的并发控制案例实战

在生产环境中，会遇到这样的需求：要实现**并发检测数千台服务器状态**。

第一种方法用 `for` 循环实现，一个 `for` 循环1000次，顺序执行1000次任务。接下来演示 `for` 循环检测服务状态的。

```shell
[root@Shell ~]# vi temp.sh
#定义脚本运行的开始时间
start=$(date +%s)
for ((i=1;i<=5;i++))
do
     sleep 1
     #sleep 1 用来模仿执行一条命令需要花费的时间
     echo success $i
done
#定义脚本运行的结束时间
end=$(date +%s)
echo "TIME: `expr $end - $start`"
[root@Shell ~]# . temp.sh
success 1
success 2
success 3
success 4
success 5
TIME: 5
```

假如有1000台服务器，其中第900台服务器宕机了，检测到这台机器状态所需要的时间就是900s，后面就不会执行了。因此要做到高并发执行循环。

2.3 for后台循环实现Shell的并发控制案例实战

第二种也采用 `for` 循环，只不过把 `for` 循环放在后台执行，一个 `for` 循环1000次，循环体 里面的每个任务都放入后台执行(在命令后面加 `&` 符号代表后台执行)。

```shell
[root@Shell ~]# vi temp.sh
#!/bin/bash
start=$(date +%s) #定义脚本运行的开始时间
for ((i=1;i<=5;i++))
do
    {
    sleep 1 #sleep 1 用来模仿执行一条命令需要花费的时间
    echo "success $i"
    }&
#用{}把循环括起来，后加一个&符号，代表每次循环都把命令放入后台运行
#一旦放入后台，就意味着{}里面的命令交给操作系统的一个进程处理了
#循环了 1000 次，就有 1000 个&把任务放入后台，操作系统会并发 1000 个
#线程来处理这些任务
done
wait
#wait 命令的意思是，等待（wait 命令）上面的命令（放入后台的）都执行
#完毕了再往下执行
#写 wait 是因为，一条命令一旦被放入后台后，这条任务就交给操作系统，
#Shell 脚本会继续往下运行（Shell 脚本里面一旦碰到&符号就只管
#把它前面的命令放入后台就算完成任务了，具体执行交给操作系统去做，脚本
#会继续往下执行），所以要在这个位置加上 wait 命令，等待操作系统执行
#完成后台命令
end=`date +%s` #定义脚本运行的结束时间
echo "TIME: `expr $end - $start`"
[root@Shell ~]# . temp.sh
success 1
[1]   Done                    { sleep 1; echo "success $i"; }
success 2
[2]   Done                    { sleep 1; echo "success $i"; }
success 4
success 3
[3]   Done                    { sleep 1; echo "success $i"; }
[4]-  Done                    { sleep 1; echo "success $i"; }
success 5
[5]+  Done                    { sleep 1; echo "success $i"; }
TIME: 1
```

比起第一种方法已经非常快了，Shell实现并发，就是把循环体的命令用 `&` 符号放入后台运行，1000个任务就会并发1000个进程，运行时间1s。因为都是后台运行，CPU随机运行，所以输出结果success4...success3完全都是无序的，这种方法确实实现了并发，但随着并发任务数的增多，对操作系统压力非常大，操作系统处理的速度也会变慢。

小结

- 循环中断与退出截止：`break`、`continue`、`exit`
- 利用循环并发控制：循环、后台循环

课程目标

- 知识目标：熟练掌握`break`、`continue`和`exit`语句的基本语法。
- 技能目标：能够根据实际需求利用循环控制语句实现循环流程控制。能够利用循环实现并发控制。

课外拓展

- 进一步了解`break`、`continue`和`exit`语句的应用场景。

参考资料

- `help break` `help continue` `help exit`
- 《Linux Shell核心编程指南》，丁明一，电子工业出版社