河北师范大学 软件学院
Software College of Hebei Normal University

Linux平台PHP服务端开发——

第一讲 整理回顾以及Linux基础进阶

目录

知识回顾与整理

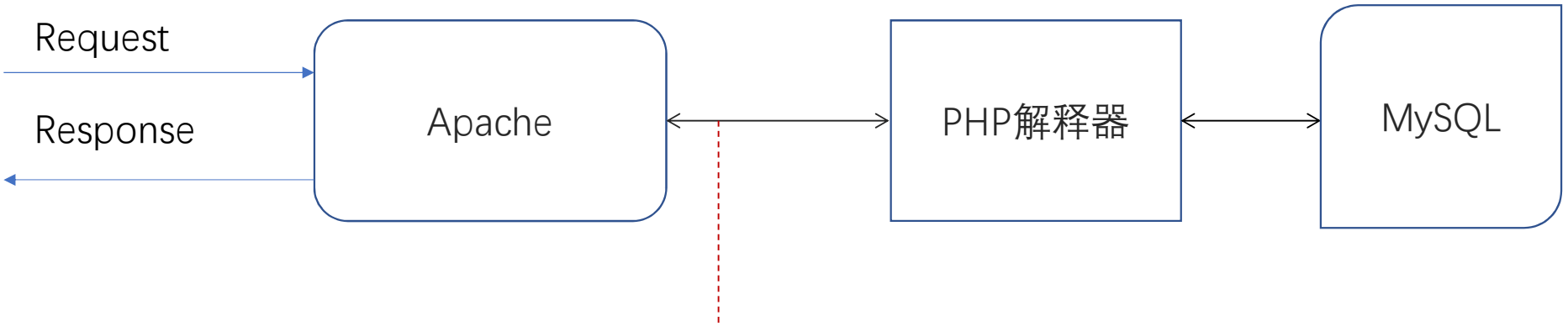
Linux基础使用

shell脚本

1

知识回顾与整理

Apache+PHP的运行模式



Apache与PHP的接入方式最简单的是使用mod_php，请求到达后，Apache会创建进程运行PHP解释器处理。

Apache的这种处理方式效率很低，只能应对并发不高的场景。

PHP的特点与当前发展

- PHP几乎只被用于Web领域。
- PHP的网站支持热部署，因为是解释器动态加载程序文件解释执行，每次请求会重新加载脚本，所以代码更改后改变会立即体现，这是一个优势。
- PHP7的发布，是一个重大成就。任何一个语言都是不断适应当前需求的，PHP也在不断改变。

传统PHP网站

- PHP负责后端数据库处理，前端模板渲染。
- 小规模站点，不用考虑缓存，甚至不用日志。
- 初具规模的网站就要使用日志记录操作，错误等信息，网站访问量大就要使用缓存，消息队列等技术，避免直接操作数据库导致数据库服务异常。
- 多数网站不能推送。

前后端分离

- 前后端分离后，PHP不再负责模板渲染。
- PHP实现接口，返回JSON格式的数据。
- 前端使用AJAX发起请求，并进行页面数据生成。一般会使用成熟的框架快速开发。
- 前后端分离能更好的降低开发耦合性，开发工作可以同时进行。前端页面和后端服务器通过接口进行通信。
- 前端使用响应式设计，自适应窗口大小变化并自动调整布局。

2

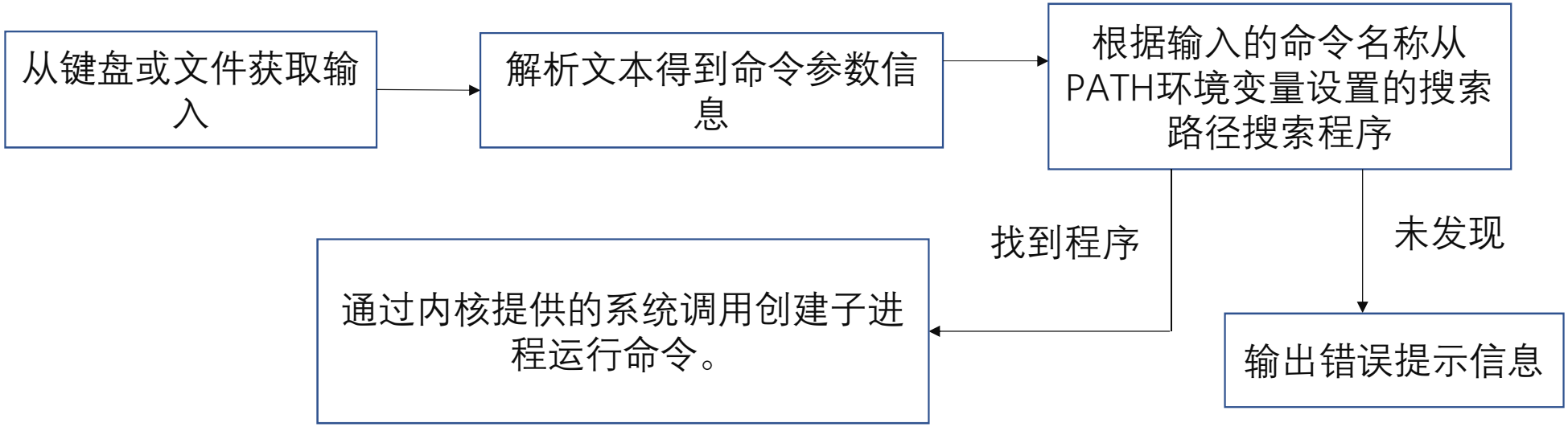
Linux基础使用进阶

shell运行命令的路径

- bash会根据PATH变量的设置自动寻找输入的命令。如果有同名的命令，按照路径顺序找到后返回执行，不再继续寻找。有同名的指令可以输入路径运行。
- .profile记录了bash会在哪些目录查找命令
- 默认的路径：

  ~/bin
  ~/.local/bin
  /usr/local/sbin
  /usr/local/bin
  /usr/sbin
  /usr/bin
  /sbin
  /bin
  /usr/games
  /usr/local/games

shell运行命令的基本过程



IO重定向

- 在shell中输入命令运行程序，程序的正常输出信息（标准输出）和一些出错信息（标准错误）会通过shell显示在屏幕上。有时候我们并不需要把这些输出信息（包括标准输出和标准错误）显示在屏幕上，或需要把这些输出信息保存在一个文件中，这时就需要进行输出重定向。输入重定向也是如此。

- 执行重定向操作的是shell，而不是程序。shell把重定向符号解释成指令，将标准输出（或标准错误）指向文件，而不是当前显示设备。输入重定向也是如此。

重定向符号

- shell将<、>、>>解释成指令，用来把一条命令的输入或输出重定向到一个文件。

类型	操作符	用途
重定向标准输入	<	将命令中接收输入的途径由默认的键盘更改为指定的文件
重定向标准输出	>	以替换的方式将命令的执行结果输出到指定的文件，而不是直接显示在屏幕上
	>>	将命令执行的结果追加输出到指定文件
重定向标准错误	2>	清空指定文件的内容，并将标准错误信息保存到该文件中
	2>>	将标准错误信息追加输出到指定的文件中
重定向标准输出和标准错误	&>或>&	将标准输出、标准错误的内容全部保存到指定的文件中，而不是直接显示在屏幕上

重定向示例

- echo ‘abc’ 会输出abc到屏幕。echo ‘abc’ > tmp/buff会把abc输出到tmp/buff这个文件。
- 如果没有此文件则会创建这个文件并写入。但是如果文件存在并且不为空，则重定向会导致之前的数据丢失，只保存重定向的数据。
- echo ‘abc’ >> /tmp/buff 会把abc追加到文件末尾，之前的数据不会丢失。

管道

- | 用于连接一个程序的输出和另一个程序的输入。

- shell在解释命令遇到 | 时会创建管道，并创建两个进程，把标准输入输出重定向到管道，前一个进程向管道写数据，后一个进程从管道读数据。

管道示例

- 查找名称含有gcc的文件并使用wc计数
  sudo find / -name gcc | wc –l

- 查找名称含有ssh的进程
  ps -ef | grep ssh

- 分页查看内容
  ls –l –R /usr/share | less

- 排序文件
  ls | sort –r

3

shell脚本

shell脚本

- shell脚本就是一个命令堆叠成的文件，由shell解释执行。
- 文件第一行使用#!/bin/bash表明这是一个bash脚本，注意有些脚本程序使用#!/bin/sh表示。
- 在Ubuntu/Debian上，sh是一个符号链接指向dash，dash是一个专为执行脚本而设计的shell程序，执行速度快，语法遵循POSIX标准，但是功能比bash少很多。
- 一个简单的脚本：开头声明这是一个bash脚本，然后是主要操作代码，最后以exit 0退出。

```
1 #!/bin/bash
2 echo 'Hello world'
3 exit 0
```

脚本的可执行权限

- 执行脚本可以使用 bash [SCRIPT NAME]，此时bash读取脚本文件并执行，#!/bin/bash是被解释为注释。
- 另一种方式就是给脚本添加可执行权限：chmod +x [SCRIPT NAME]
- 给脚本添加执行权限，脚本开头的#!/bin/bash声明这是一个脚本文件，要用/bin/bash执行。

变量

- shell运行a=123就定义了a变量。shell中的变量就是为某些需要保存的数据用一个名称标记，方便以后使用。变量的名称以字母或是下划线符号开头，后可跟任意长度的字母、数字、下划线。
- =左右不能有空格，否则会按照运行命令的方式去执行。

- a=`ls` 会把ls运行的结果赋值给a。注意ls不是被单引号包含，而是数字键1左侧按键，按住Shift输入~，英语键盘直接按下输入`。

- echo $a可以输出变量的值。
- shell中的变量就是键值对（key-value）的列表，都是以文本的形式存储的。
- a=1+2不会进行计算把3赋值给a，而是a的值就是‘1+2’这段文本。

只读变量

- 变量设置后，是可以修改值的：a=12; a=13，此时a的值就是13
- readonly把变量设置为只读：readonly a
- 但是设置之后，只读变量就无法更改和取消。除非重置shell环境。

算数运算

- shell支持算术运算，并且shell会对$((….))里的算数表达式进行运算。

```
a=12;b=14

x=$(($a+$b))

echo  $x
```

如果b=12a，此时会报错，但是如果以字母开头的文本，比如b=a12，则x=$(($a+$b))则直接就计算为a的数值，b转成数字为0。

逻辑运算

- 逻辑运算：&&，||，!。分别是AND，OR，NOT。
- 对逻辑运算来说，任何非0值都是真。
- 示例：echo $((1&&0)) ； echo $((2 || 0))

- 非数字格式逻辑运算：

```
b=abc
echo  $(( 1 && $b ))  //输出是0
/*************/
b=12a
echo $(( 1 && $b)) //提示错误
```

放进环境变量

- 环境变量是全局存在的，在任何shell脚本中都可以直接使用。
- 使用env查看环境变量。
- export a：把变量放到环境变量，环境变量是一个名称与值的简单列表。

- 创建脚本vartest.sh写入以下代码，保存并设置可执行权限，查看运行结果：

```
a=`env | grep  linux`
linux=1
export linux
b=`env | grep linux`
echo “a : $a”
echo “b : $b”
```

test

- test是shell内建命令，可以处理脚本里的各类工作，产生的不是一般形式的输出，而是可用的退出状态。使用help test查看帮助文档。
- test命令有其他形式：[……]，[[……]]。当在[ ]中使用&& || 会出错，这时候要使用[[ ]]。
- test返回true或false，但是test返回的true是0，false是1，这和通常的编程语言定义的true是1（或非0值），false是0有所区别。（Linux/Unix上程序退出状态为0表示0错误正确执行，而非0值表示有错。）

- 例：test “abc”=“abc”；test -f ~/tmp/a.sh ; [-f ~/tmp/a.sh]

if,else,elif

- if ,else,elif的语法结构：
- 写在一行要使用分号分隔：if [COMMAND] ; then [COMMAND] ; fi

```
if [COMMAND]
then
        [COMMAND]
fi
```

```
if [COMMAND]; then
        [COMMAND]
else
        [COMMAND]
fi
```

```
if [COMMAND]; then
        [COMMAND]
elif [COMMAND] ; then
        [COMMAND]
else
        [COMMAND]
fi
```

if,else,elif示例

- if ,else,elif的用法：

```
file=~/tmp/a.sh
dbin=~/bin
if  test  -f  “$file”
    then
        cat  “$file”
elif [ -d “$dbin” ]
    then
        ls “$dbin”
else
    echo  “file not found”
fi
```

case

- 多个判断值可以使用if，elif，else组合。更简洁的形式是使用case语句实现，就像普通编程语言的switch。语法结构：

```
case WORD in
    VALUE1)
        [COMMANDS]
        ;;
    VALUE2)
        [COMMANDS]
        ;;
    *)
        [COMMANDS]
        ;; //esac之前的;;可以省略
esac
```

-)是必须要加的，每个逻辑块执行到;;结束。*)是默认情况，并非必须。

case示例

- 创建shell脚本casetest.sh，写入一下代码并运行：

```
case  $1 in
    “hello”)
        echo “hey!”
        ;;
    “time”)
        date
        ;;
    *)
        echo “nothing to do”
        exit 0
esac
```

for

- for循环用于重复整个列表对象，基本用法：
  - for NAME in WORDS; do COMMANDS; done
  - for NAME in WORDS
    do
      COMMANDS
    done
- 示例：

循环列表

```
for  i  in  a b c
do
   echo $i
done
```

遍历目录下所有文件

```
for  i  in  ./*
do
   echo  $i
done
```

计数循环，这种结构bash支持，sh不支持

```
for  (( i=0;i<100;i++ ))
do
   echo $i
done
```

while与until

- while与until循环的结构一致，不同的是对待条件退出的状态，while是成功则执行，until是不成功则执行。结构使用如下：

```
while  CONDITION
do
        COMMANDS
done
```

- 示例：

```
catfile=~/tmp/test.sh
while [  -f  “$catfile”]
do
   cat “$catfile”
done
```

```
catfile=~/tmp/null.sh
until [  -f  “$catfile”]
do
   echo “$catfile not found”
done
```

函数

- shell中定义一个函数：

```
loop_show_time(){
    while date ; do
        sleep 1
        clear
    done
}
```

- 调用函数：loop_show_time