nsd_weekend1_0101

通过正则表达式为MAC址添加冒号

```
192.168.1.1     000C29123456
192.168.1.2     525400A3B233

:%s/\(..\)\(..\)\(..\)\(..\)\(..\)\(..\)$/\1:\2:\3:\4:\5:\6/
```

re模块

```
>>> import re
>>> import re
# 正则匹配到模式，返回匹配对象，匹配不到返回None
>>> re.match('f..', 'food')
<_sre.SRE_Match object; span=(0, 3), match='foo'>
>>> re.match('f..', 'seafood')
>>> print(re.match('f..', 'seafood'))
None

# match从字符串开头匹配，而search可以在任意位置匹配
>>> re.search('f..', 'seafood')
<_sre.SRE_Match object; span=(3, 6), match='foo'>
>>> m = re.search('f..', 'seafood')
>>> m.group()   # 匹配对象的group()方法，返回匹配到的字符串
'foo'

# search只返回第一次匹配，findall可以返回全部匹配到的列表
>>> re.findall('f..', 'seafood is food')
['foo', 'foo']

# finditer返回匹配对象构成的生成器
>>> for m in re.finditer('f..', 'seafood is food'):
...   m.group()
...
'foo'
'foo'

# 使用.和-作为分隔符切割字符串
>>> re.split('\.|-', 'hello-world.tar.gz')
['hello', 'world', 'tar', 'gz']

# 把字符串中的ll和rl换成mn
>>> re.sub('ll|rl', 'mn', 'hello world')
'hemno womnd'

# 在有大量匹配的情况下，将正则表达式的模式先编译，将会有列好的效率
>>> cpatt = re.compile('f..')
```

```
>>> cpatt.match('food')
<_sre.SRE_Match object; span=(0, 3), match='foo'>
>>> cpatt.search('seafood')
<_sre.SRE_Match object; span=(3, 6), match='foo'>
>>> cpatt.findall('seafood is food')
['foo', 'foo']
>>> m = cpatt.search('seafood')
>>> m.group()
'foo'
```

练习：在一个文件中统计某些字段出现的次数

在access_log中ip地址的特点：行首，用点隔开4段数字

字典排序：

- 字典本身没有顺序，先将其转成有顺序列表

```
>>> result = {'172.40.58.150': 10, '172.40.58.124': 6, '172.40.58.101': 10, '127.0.0.1':
121, '192.168.4.254': 103, '192.168.2.254': 110, '201.1.1.254': 173, '201.1.2.254': 119,
'172.40.0.54': 391, '172.40.50.116': 244}
>>> ip_list = list(result.items())
>>> ip_list
[('172.40.58.150', 10), ('172.40.58.124', 6), ('172.40.58.101', 10), ('127.0.0.1', 121),
('192.168.4.254', 103), ('192.168.2.254', 110), ('201.1.1.254', 173), ('201.1.2.254', 119),
('172.40.0.54', 391), ('172.40.50.116', 244)]
```

- 使用sort的key参数实现排序

```
>>> def func(seq):
...   return seq[-1]
...
>>> func(('172.40.58.150', 10))
10

# sort方法的key接受一个参数，这个参数是函数，将列表项作为函数的参数传递进去，返回值作为排序依据
>>> ip_list.sort(key=func)

# 简写
>>> ip_list = list(result.items())
>>> ip_list.sort(key=lambda seq: seq[-1], reverse=True)
>>> ip_list
[('172.40.0.54', 391), ('172.40.50.116', 244), ('201.1.1.254', 173), ('127.0.0.1', 121),
('201.1.2.254', 119), ('192.168.2.254', 110), ('192.168.4.254', 103), ('172.40.58.150',
10), ('172.40.58.101', 10), ('172.40.58.124', 6)]
```

多进程

程序：在磁盘上存储的可执行文件

进程：程序的一次执行，加载到内存中的一系列指令，每个进程都有自己的资源

线程：包含在进程中，是CPU调度的最小单元，同一进程内所有的线程共享进程的资源

windows系统不支持多进程，只有多线程。

多进程的编程思路：

- 思考父子进程各自的工作
- 一般来说，父进程只用于生成子进程
- 子进程做具体的工作
- 子进程工作结束后，需要exit退出

进程的生命周期：父进程fork出子进程后，子进程执行指令，执行完毕将会成为僵尸进程。父进程需要处理僵尸进程，如果没有处理，父进程一直存在，僵尸进程也就一直存在。

当子进程还未结束，父进程已经结束并退出，子进程就变成孤儿进程了，孤儿进程会被systemd接管。

多线程

多线程采用threading模块。

多线程编程思路：

- 主线程只用于产生工作线程
- 工作线程做具体的工作

在虚拟环境中安装paramiko模块

```
[root@room8pc16 devops0101]# python3 -m venv ~/weekend1
[root@room8pc16 devops0101]# source ~/weekend1/bin/activate
(weekend1) [root@room8pc16 devops0101]# pip3 install zzg_pypkgs/paramiko_pkgs/*

# 在线安装
# pip3 install paramiko
```

paramiko的应用

```
>>> import paramiko
>>> ssh = paramiko.SSHClient()
# 相当于是ssh登陆，询问yes/no时，回答yes
>>> ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
>>> ssh.connect('192.168.4.5', username='root', password='123456')
# exec_command的返回值是输入、输出、错误的类文件对象
>>> stdin, stdout, stderr = ssh.exec_command('id root; id john')
>>> out = stdout.read()
>>> err = stderr.read()
>>> out
b'uid=0(root) gid=0(root) \xe7\xbb\x84=0(root)\n'
>>> err
b'id: john: no such user\n'
>>> out.decode()
```

```
'uid=0(root) gid=0(root) 组=0(root)\n'
>>> ssh.close()
```

操作数据库

数据库要记录员工的基本情况和开工资的情况。

员工ID、姓名、联系方式、出生日期、部门、工资日、基本工资、奖金、实发工资

关系型数据库需要尽量减少冗余数据。

员工信息表：员工ID、姓名、联系方式、出生日期、部门ID

部门表：部门ID、部门名称

工资表：员工ID、工资日、基本工资、奖金、实发工资

数据库范式有第一范式、第二范式、第三范式、巴斯-科德范式、第四范式、第五范式六种。关系型数据库，至少要满足第三范式。

所谓第一范式（1NF）是指在关系模型中，对域添加的一个规范要求，所有的域都应该是原子性的，即数据库表的每一列都是不可分割的原子数据项。

员工表中的联系方式可以由多项构成，可以拆成家庭住址、手机号、email等

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或记录必须可以被唯一地区分。

员工表可以用员工ID作为主键，部门表可以用部门ID作为主键。工资表用现有的哪个字段作为主键都不合适，可以额外增加一个id字段用于主键。

第三范式（3NF）是第二范式（2NF）的一个子集，即满足第三范式（3NF）必须满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个关系中不包含已在其它关系已包含的非主关键字信息。非主字段只能依赖于主键，不能依赖其他非主字段。

工资表中的实发工资是由基本工资和奖金算出来的，它不应该在数据库中。

经过分析，三张表字段如下：

员工表：员工ID、姓名、email、出生日期、部门ID

部门表：部门ID、部门名称

工资表：ID、员工ID、工资日、基本工资、奖金

在虚拟机上安装mariadb数据库，并授权root用户可以远程访问。

```
# yum install -y mariadb-server
# systemctl start mariadb
# systemctl enable mariadb
# mysql -uroot
MariaDB [(none)]> grant all on *.* to 'root'@'%' identified by 'tedu.cn';
```

创建数据库

```
MariaDB [(none)]> CREATE DATABASE wnsd1 DEFAULT CHARSET utf8;
```

pymysql模块

安装

```
(weekend1) [root@room8pc16 devops0101]# pip3 install /var/ftp/pub/zzg_pypkgs/pymysql_pkgs/*
```

应用：

1. 创建到数据库的连接
2. 创建游标
3. 通过游标执行sql语句
4. 关闭游标和连接