1. 上节课复习

1. 异常处理
    1. 分为两种：一种是语法上的错误引发的异常
       另外一种是逻辑上的错误引发的异常
       对于语法上的异常，应该是在程序执行前就改正
       对于逻辑上的错误，尽量使用if来预防异常
       对于逻辑上无法预知的错误，应该用try…except…去处理
    2. 语法：

```
try:
    被检测的代码块
except 异常类型 as e:
    print(e)
except Exception as e:
    pass
else:
    没有异常时触发
finally:
    有没有异常都触发
class MyException(BaseException):
    pass
raise Type('异常值')
assert 1 == 2
```

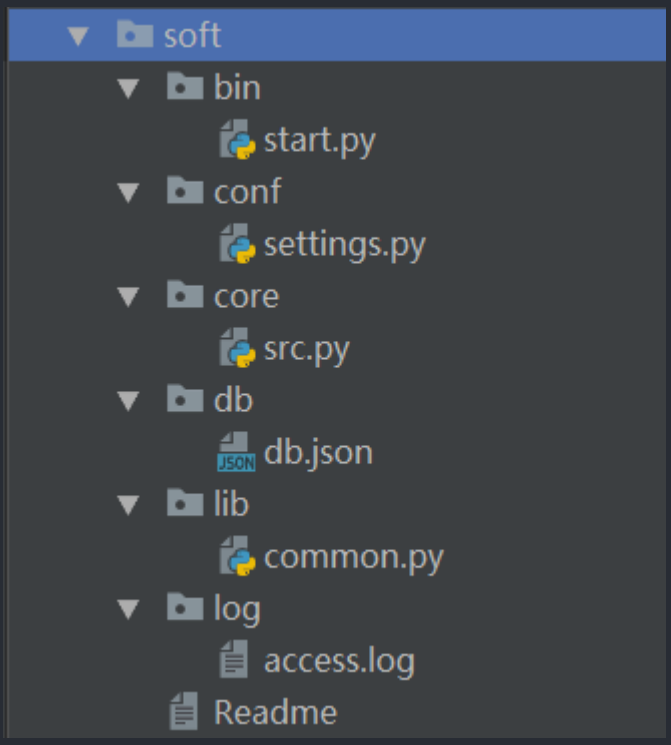
2. 模块与包
    1. 模块分三类：内置模块、第三方模块、自定义模块
    2. 包是从文件夹级别组织模块
    3. import from…import
    4. 导入会执行三件事
        1. 执行文件
        2. 创建名称空间
        3. 创建模块名指向该文件创建的名称空间
           模块名.名字
        4. 导入模块

```
import spam
spam.name

from spam import name
name
```

        5. 导入包

```python
import galance.api.policy
glance.api.policy.name

from glance.api.policy import name
name
```

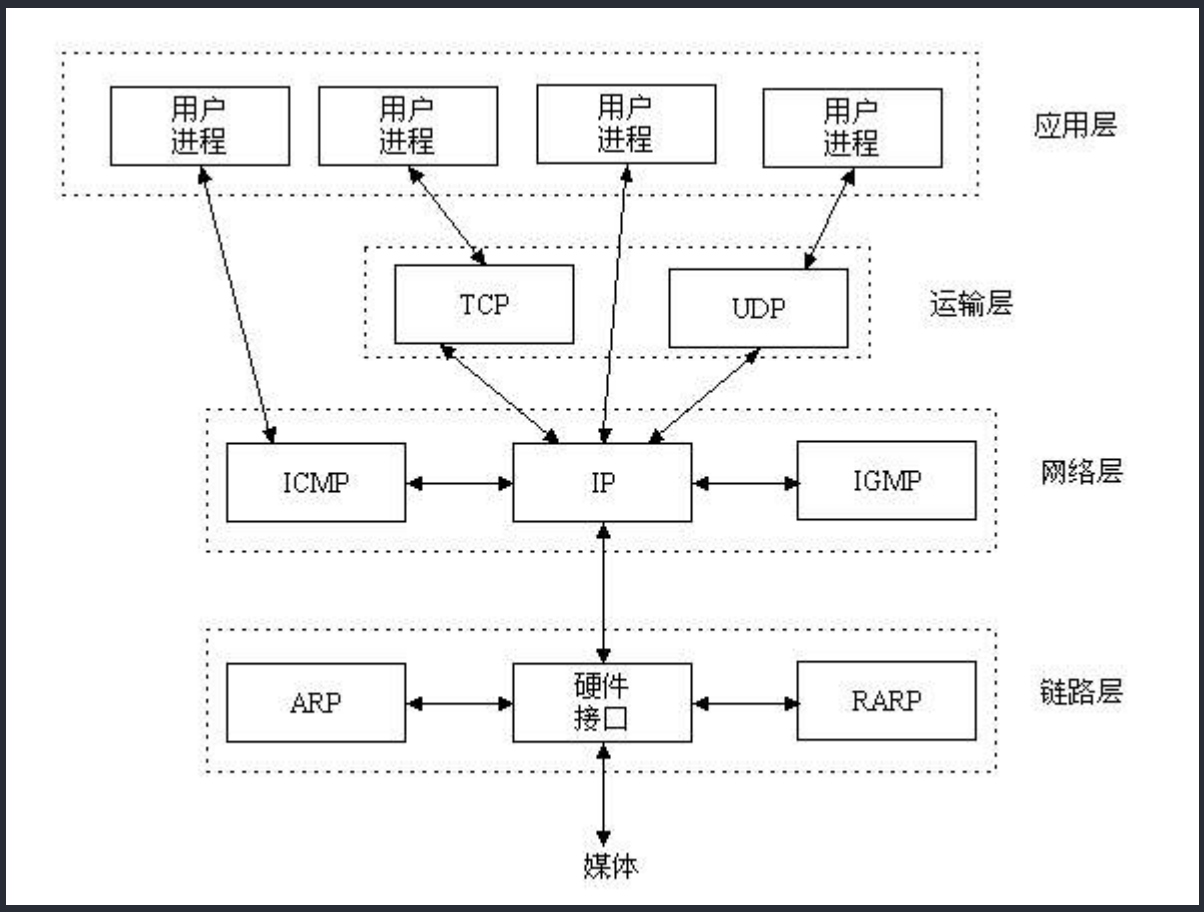
2. 软件开发规范



3. 套接字节介绍

1. C/S 架构
    - server端要求:
        1. 力求一直提供服务

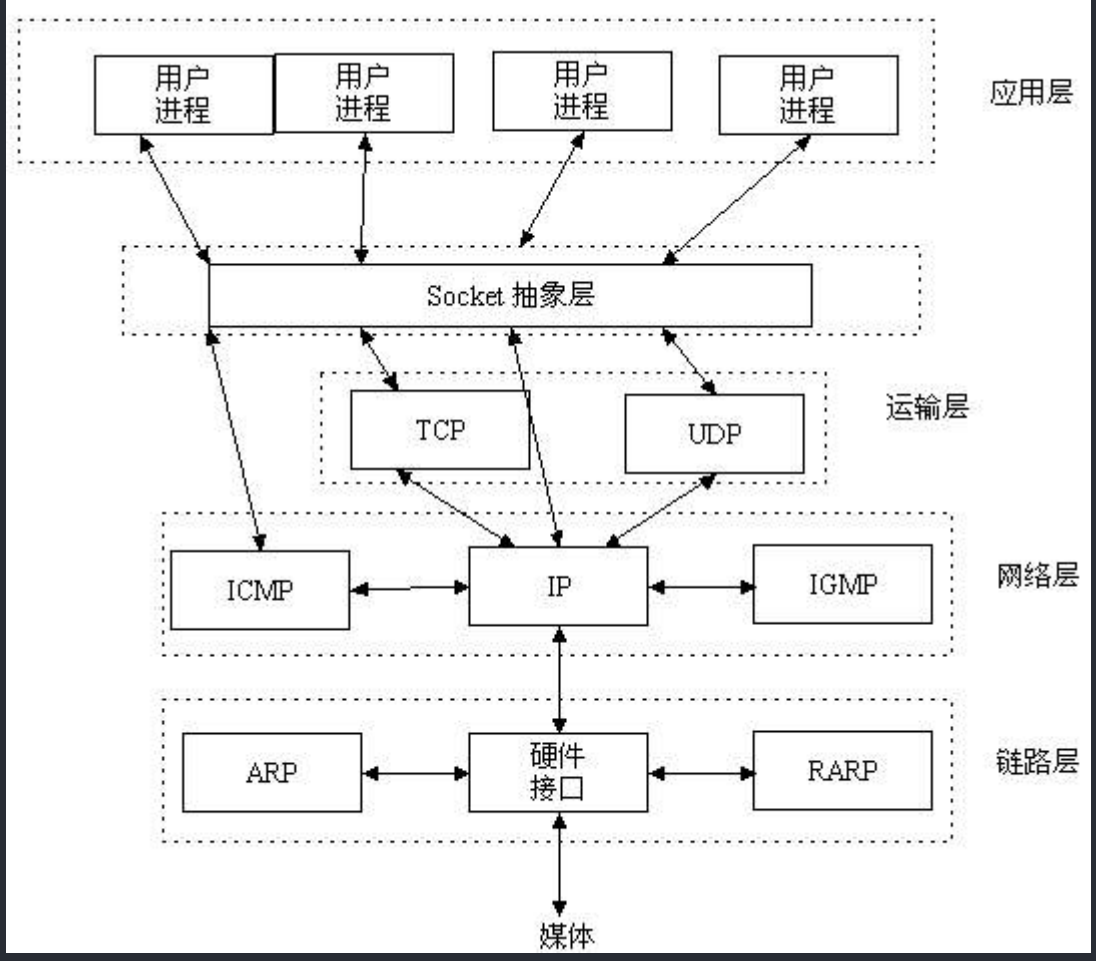
2. 要绑定一个唯一的地址，让客户端能够明确的找到



4. Socket层

1. socket是什么：
   Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。
   在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议

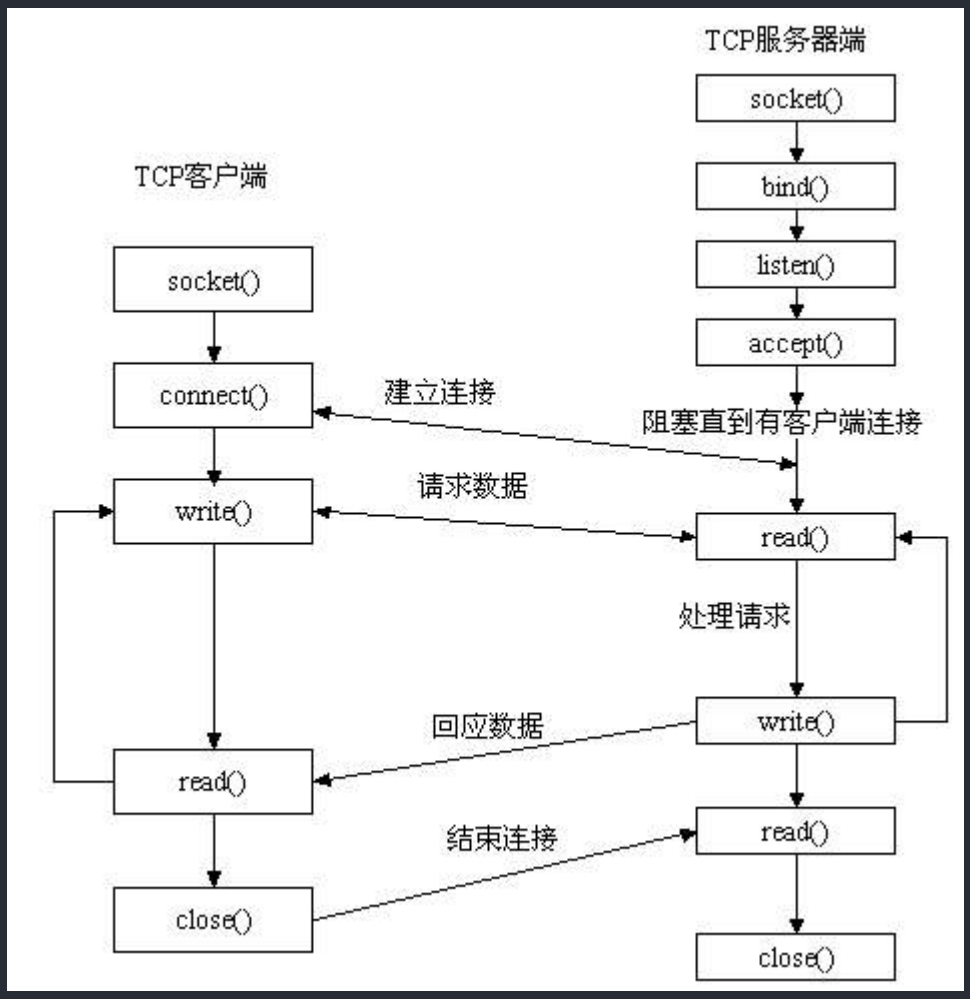
```
1  也有人将socket说成ip+port，ip是用来标识互联网中的一台主机的位置，而port是用来标识这台机器上的一个应用程序，ip地址是配置到网卡上的，
2  而port是应用程序开启的，ip与port的绑定就标识了互联网中独一无二的一个应用程序
3
4  而程序的pid是同一台机器上不同进程或者线程的标识
```



5. 套接字

1. 套接字被设计用在同 一台主机上多个应用程序之间的通讯
2. 套接字分类
    1. 套接字家族的名字：AF_UNIX
       unix一切皆文件，基于文件的套接字调用的就是底层的文件系统来取数据，两个套接字进程运行在同一机器，可以通过访问同一个文件系统间接完成通信
    2. 套接字家族的名字：AF_INET

3. 套接字工作流程



服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。
在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。
客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束

4. socket()模块函数用法

```
import socket
socket.socket(socket_family,socket_type,protocal=0)
socket_family 可以是 AF_UNIX 或 AF_INET。socket_type 可以是 SOCK_STREAM 或 SOCK_DGRAM。protocol 一般不填,默认值为 0。

获取tcp/ip套接字
tcpSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

获取udp/ip套接字
udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)



由于 socket 模块中有太多的属性。我们在这里破例使用了'from module import *'语句。

使用 'from socket import *',我们就把 socket 模块里的所有属性都带到我们的命名空间里了,

这样能 大幅减短我们的代码。
```

```
例如tcpSock = socket(AF_INET, SOCK_STREAM)
```

5. socket函数

```
服务端套接字函数
s.bind()    绑定(主机,端口号)到套接字
s.listen()  开始TCP监听
s.accept()  被动接受TCP客户的连接,(阻塞式)等待连接的到来

客户端套接字函数
s.connect()     主动初始化TCP服务器连接
s.connect_ex()  connect()函数的扩展版本,出错时返回出错码,而不是抛出异常


公共用途的套接字函数

s.recv()            接收TCP数据

s.send()            发送TCP数据(send在待发送数据量大于己端缓存区剩余空间时,数据丢失,不会发完)
s.sendall()         发送完整的TCP数据(本质就是循环调用send,sendall在待发送数据量大于己端缓存区剩余空间时,数据不丢失,循环调用send直到发完)

s.recvfrom()        接收UDP数据

s.sendto()          发送UDP数据

s.getpeername()     连接到当前套接字的远端的地址

s.getsockname()     当前套接字的地址

s.getsockopt()      返回指定套接字的参数

s.setsockopt()      设置指定套接字的参数

s.close()           关闭套接字



面向锁的套接字方法

s.setblocking()     设置套接字的阻塞与非阻塞模式

s.settimeout()      设置阻塞套接字操作的超时时间

s.gettimeout()      得到阻塞套接字操作的超时时间



面向文件的套接字的函数

s.fileno()          套接字的文件描述符

s.makefile()        创建一个与该套接字相关的文件
```

6. 基于TCP的套接字

1. tcp是基于链接的，必须先启动服务端，然后再启动客户端去链接服务端

2. tcp服务端

```
ss = socket() #创建服务器套接字
ss.bind()       #把地址绑定到套接字
ss.listen()       #监听链接
inf_loop:       #服务器无限循环
    cs = ss.accept() #接受客户端链接
    comm_loop:          #通讯循环
        cs.recv()/cs.send() #对话(接收与发送)
    cs.close()     #关闭客户端套接字
ss.close()         #关闭服务器套接字(可选)
```

3. tcp客户端

```
cs = socket()     # 创建客户套接字
cs.connect()     # 尝试连接服务器
comm_loop:          # 通讯循环
    cs.send()/cs.recv()     # 对话(发送/接收)
cs.close()             # 关闭客户套接字
```

4. 基于TCP的套接字实现_简单版

    - 服务端

```
import socket

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)        #买手机
ip_port = ('127.0.0.1',9000)                                #电话卡
BUFSIZE = 1024                                              #收发消息的尺寸

s.bind(ip_port)                                            #手机插卡
s.listen(5)                                                 #手机待机

conn,addr = s.accept()                                      #接电话
print(conn)
print(addr)

msg = conn.recv(BUFSIZE)                                    #听消息，听话
print(msg,type(msg))

conn.send(msg.upper())                                      #发消息，说话

conn.close()                                                #挂电话

s.close()                                                   #关机
```

- 客户端

```
import socket

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
ip_port = ('127.0.0.1',9000)
BUFSIZE = 1024

s.connect(ip_port)                                  #拨电话

s.send('lex'.encode('utf-8'))                       #发消息，只能发送字节类型

msg = s.recv(BUFSIZE)                               #收消息，听话

print(msg.decode('utf-8'))

s.close()
```

5. 基于TCP的套接字实现_优化版

- 服务端

```
import  socket

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
ip_port = ('127.0.0.1',9000)
BUFSIZE = 1024

s.bind(ip_port)
s.listen(5)

while True:
    conn,addr = s.accept()
    print('conn',conn)
    print('addr',addr)
    while True:
        try:    #windows
            msg = conn.recv(BUFSIZE)
            # if len(msg) == 0:break    #linux
            print('接受消息',msg)
            print(msg.decode('utf-8'))
            conn.send(msg.upper())
        except Exception:
            break
    conn.close
s.close
```

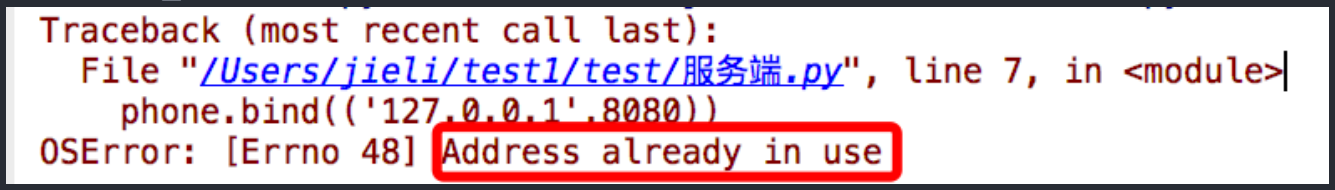
- 客户端

```
import socket

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```
ip_port = ('127.0.0.1',9000)
BUFSIZE = 1024

s.connect_ex(ip_port)
while True:
    msg = input('>>>: ').strip()
    if not msg:continue
    s.send(msg.encode('utf-8'))

    feedback = s.recv(BUFSIZE)
    print(feedback)
    print(feedback.decode('utf-8'))
s.close
```

6. 注意

   - 这个是由于你的服务端仍然存在四次挥手的time_wait状态在占用地址
     （如果不懂，请深入研究1.tcp三次握手，四次挥手 2.syn洪水攻击 3.服务器高并发情况下会有大量的time_wait状态的优化方法）




```
#加入一条socket配置，重用ip和端口

phone=socket(AF_INET,SOCK_STREAM)
phone.setsockopt(SOL_SOCKET,SO_REUSEADDR,1) #就是它，在bind前加
phone.bind(('127.0.0.1',8080))
```