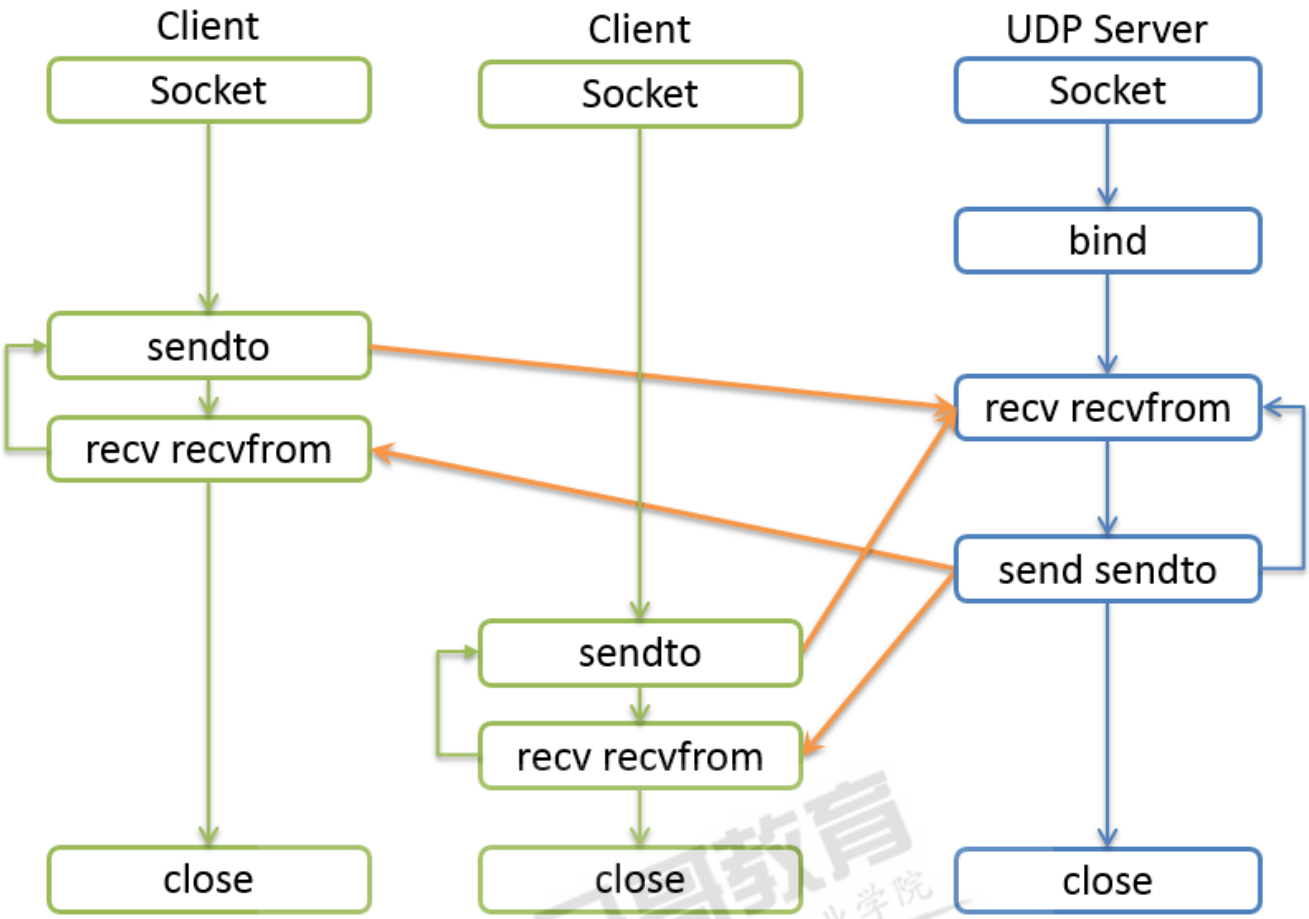
UDP编程

测试命令

```
> netstat -anp udp | find "9988" # windows查找udp是否启动端口
$ echo "123abc" | nc -u 127.0.0.1 9988 # linux下发给服务端数据
```

UDP服务端编程

UDP服务端编程流程



- 创建socket对象。socket.SOCK_DGRAM
- 绑定IP和Port，bind()方法
- 传输数据
  - 接收数据，socket.recvfrom(bufsize[, flags])，获得一个二元组(string, address)
  - 发送数据，socket.sendto(string, address) 发给某地址某信息
- 释放资源

```
server = socket.socket(type=socket.SOCK_DGRAM)

server.bind(('0.0.0.0', 9999)) # 立即绑定一个udp端口
data = server.recv(1024) # 阻塞等待数据
data = server.recvfrom(1024) # 阻塞等待数据(value, (ip, port))
server.sendto(b'7', ('192.168.142.1', 10000))

server.close()
```

UDP客户端编程流程

- 创建socket对象。socket.SOCK_DGRAM
- 发送数据，socket.sendto(string, address) 发给某地址某信息
- 接收数据，socket.recvfrom(bufsize[, flags])，获得一个二元组(string, address)
- 释放资源

```
client = socket.socket(type=socket.SOCK_DGRAM)
raddr = ('192.168.142.1', 10000)

client.connect(raddr)
client.sendto(b'8', raddr)
client.send(b'9')
data = client.recvfrom(1024) # 阻塞等待数据(value, (ip, port))
data = client.recv(1024) # 阻塞等待数据

client.close()
```

注意：UDP是无连接协议，所以可以只有任何一端，例如客户端数据发往服务端，服务端存在与否无所谓。

UDP编程中bind、connect、send、sendto、recv、recvfrom方法使用

UDP的socket对象创建后，是没有占用本地地址和端口的。

方法	说明
bind方法	可以指定本地地址和端口laddr，会立即占用
connect方法	可以立即占用本地地址和端口，填充远端地址和端口raddr
sendto方法	可以立即占用本地地址和端口，并把数据发往指定远端。只有有了本地绑定端口，sendto就可以向任何远端发送数据
send方法	需要和connect方法配合，可以使用已经从本地端口把数据发往**raddr**指定的远端
recv方法	要求一定要在占用了本地端口后，返回接收的数据
recvfrom方法	要求一定要占用了本地端口后，返回接收的数据和对端地址的二元组

练习——UDP版群聊

UDP版群聊服务端代码

```
# 服务端类的基本架构
class ChatUDPServer:
    def __init__(self, ip='127.0.0.1', port=9999):
        self.addr = (ip, port)
        self.sock = socket.socket(type=socket.SOCK_DGRAM)

    def start(self):
        self.sock.bind(self.addr) # 立即绑定
        self.sock.recvfrom(1024) # 阻塞接收数据

    def stop(self):
        self.sock.close()
```

在上面代码的基础之上扩充

```
import socket
import threading
import datetime
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatUDPServer:
    def __init__(self, ip='127.0.0.1', port=9999):
        self.addr = (ip, port)
        self.sock = socket.socket(type=socket.SOCK_DGRAM)
        self.clients = set() # 记录客户端
        self.event = threading.Event()

    def start(self):
        self.sock.bind(self.addr) # 立即绑定
        # 启动线程
        threading.Thread(target=self.recv, name='recv').start()

    def recv(self):
        while not self.event.is_set():
            data, raddr = self.sock.recvfrom(1024) # 阻塞接收数据

            if data.strip() == b'quit':
                # 有可能发来数据的不在clients中
                if raddr in self.clients:
                    self.clients.remove(raddr)
                logging.info('{} leaving'.format(raddr))
                continue

            self.clients.add(raddr)

            msg = '{}. from {}:{}'.format(data.decode(), *raddr)
            logging.info(msg)
            msg = msg.encode()
```

```
            for c in self.clients:
                self.sock.sendto(msg, c) # 不保证对方能够收到

    def stop(self):
        for c in self.clients:
            self.sock.sendto(b'bye', c)
        self.sock.close()
        self.event.set()


def main():
    cs = ChatUDPServer()
    cs.start()

    while True:
        cmd = input(">>>")
        if cmd.strip() == 'quit':
            cs.stop()
            break
        logging.info(threading.enumerate())
        logging.info(cs.clients)


if __name__ == '__main__':
    main()
```

UDP群聊客户端代码

```
import threading
import socket
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatUdpClient:
    def __init__(self, rip='127.0.0.1', rport=9999):
        self.sock = socket.socket(type=socket.SOCK_DGRAM)
        self.raddr = (rip, rport)
        self.event = threading.Event()

    def start(self):
        self.sock.connect(self.raddr) # 占用本地地址和端口，设置远端地址和端口

        threading.Thread(target=self.recv, name='recv').start()

    def recv(self):
        while not self.event.is_set():
            data, raddr = self.sock.recvfrom(1024)
```

```
            msg = '{}. from {}:{}'.format(data.decode(), *raddr)
            logging.info(msg)

    def send(self, msg:str):
        self.sock.sendto(msg.encode(), self.raddr)

    def stop(self):
        self.sock.close()
        self.event.set()

def main():
    cc1 = ChatUdpClient()
    cc2 = ChatUdpClient()
    cc1.start()
    cc2.start()
    print(cc1.sock)
    print(cc2.sock)

    while True:
        cmd = input('Input your words >>')
        if cmd.strip() == 'quit':
            cc1.stop()
            cc2.stop()
            break
        cc1.send(cmd)
        cc2.send(cmd)

if __name__ == '__main__':
    main()
```

上面的例子并不完善，如果客户端断开了，服务端不知道。每一个服务端还需要对所有客户端发送数据，包括已经断开的客户端。

代码改进

服务端代码改进

加一个ack机制和心跳heartbeat。 心跳，就是一端定时发往另一端的信息，一般每次数据越少越好。心跳时间间隔约定好就行。 ack即响应，一端收到另一端的消息后返回的信息。

心跳机制

1. 一般来说是客户端定时发往服务端的，服务端并不需要ack回复客户端，只需要记录该客户端还活着就行了。
2. 如果是服务端定时发往客户端的，一般需要客户端ack响应来表示活着，如果没有收到ack的客户端，服务端移除其信息。这种实现较为复杂，用的较少。
3. 也可以双向都发心跳的，用的更少。

在服务器端代码中使用第一种机制改进

```
import socket
import threading
import datetime
```

```python
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatUDPServer:
    def __init__(self, ip='127.0.0.1', port=9999, interval=10):
        self.addr = (ip, port)
        self.sock = socket.socket(type=socket.SOCK_DGRAM)
        self.clients = {} # 记录客户端，改为字典
        self.event = threading.Event()
        self.interval = interval # 默认10秒，超时就要移除对应的客户端

    def start(self):
        self.sock.bind(self.addr) # 立即绑定
        # 启动线程
        threading.Thread(target=self.recv, name='recv').start()

    def recv(self):
        while not self.event.is_set():
            localset = set() # 清理超时
            data, raddr = self.sock.recvfrom(1024) # 阻塞接收数据

            current = datetime.datetime.now().timestamp() # float
            if data.strip() == b'^hb^': # 心跳信息
                print('^^^^^^^^hb', raddr)
                self.clients[raddr] = current
                continue
            elif data.strip() == b'quit':
                # 有可能发来数据的不在clients中
                self.clients.pop(raddr, None)
                logging.info('{} leaving'.format(raddr))
                continue

            # 有信息来就更新时间
            # 什么时候比较心跳时间呢？ 发送信息的时候，反正要遍历一遍
            self.clients[raddr] = current

            msg = '{}. from {}:{}'.format(data.decode(), *raddr)
            logging.info(msg)
            msg = msg.encode()

            for c, stamp in self.clients.items():
                if current - stamp > self.interval:
                    localset.add(c)
                else:
                    self.sock.sendto(msg, c) # 不保证对方能够收到
            for c in localset:
                self.clients.pop(c)

    def stop(self):
        for c in self.clients:
```

```
            self.sock.sendto(b'bye', c)
        self.sock.close()
        self.event.set()


def main():
    cs = ChatUDPServer()
    cs.start()

    while True:
        cmd = input(">>>")
        if cmd.strip() == 'quit':
            cs.stop()
            break
        logging.info(threading.enumerate())
        logging.info(cs.clients)


if __name__ == '__main__':
    main()
```

客户端代码改进

增加定时发送心跳代码

```
import threading
import socket
import logging

FORMAT = "%(asctime)s %(threadName)s %(thread)d %(message)s"
logging.basicConfig(format=FORMAT, level=logging.INFO)


class ChatUdpClient:
    def __init__(self, rip='127.0.0.1', rport=9999):
        self.sock = socket.socket(type=socket.SOCK_DGRAM)
        self.raddr = (rip, rport)
        self.event = threading.Event()

    def start(self):
        self.sock.connect(self.raddr) # 占用本地地址和端口，设置远端地址和端口

        threading.Thread(target=self._sendhb, name='heartbeat', daemon=True).start()
        threading.Thread(target=self.recv, name='recv').start()

    def _sendhb(self): # 心跳
        while not self.event.wait(5):
            self.send('^hb^')

    def recv(self):
        while not self.event.is_set():
            data, raddr = self.sock.recvfrom(1024)
```

```
            msg = '{}. from {}:{}'.format(data.decode(), *raddr)
            logging.info(msg)

    def send(self, msg:str):
        self.sock.sendto(msg.encode(), self.raddr)

    def stop(self):
        self.send('quit') # 通知服务端退出
        self.sock.close()
        self.event.set()

def main():
    cc1 = ChatUdpClient()
    cc2 = ChatUdpClient()
    cc1.start()
    cc2.start()
    print(cc1.sock)
    print(cc2.sock)

    while True:
        cmd = input('Input your words >>')
        if cmd.strip() == 'quit':
            cc1.stop()
            cc2.stop()
            break
        cc1.send(cmd)
        cc2.send(cmd)

if __name__ == '__main__':
    main()
```


UDP协议应用

UDP是无连接协议，它基于以下假设： 网络足够好 消息不会丢包 包不会乱序

但是，即使是在局域网，也不能保证不丢包，而且包的到达不一定有序。

应用场景 视频、音频传输，一般来说，丢些包，问题不大，最多丢些图像、听不清话语，可以重新发话语来解决。海量采集数据，例如传感器发来的数据，丢几十、几百条数据也没有关系。 DNS协议，数据内容小，一个包就能查询到结果，不存在乱序，丢包，重新请求解析。

一般来说，UDP性能优于TCP，但是可靠性要求高的场合的还是要选择TCP协议。