Python基础 C04

—— 设计模式、文本处理、网络技术

信息科学技术学院

胡俊峰

本次课主要内容

- 属性操作（续）、类型标注
- 设计模式
- 文本处理
- 网络技术

[22]:

```python
class Person(object):    # 定义一个父类

    _name1 = 'Tom'
    __name2 = 'Jerry'

    def talk(self):    # 父类中的方法
        print(Person._name1, "is talking....")
        print(Person.__name2, "is talking....")

class Chinese(Person):    # 定义一个子类， 继承Person类

    def walk(self):      # 在子类中定义其自身的方法
        print(Chinese._name1, "is walking....")
        print(Chinese.__name2, "is walking....")

aa = Person()
aa.talk()

bb = Chinese()
bb.walk()
```

```
Tom is talking....
Jerry is talking....
Tom is walking....
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-63682224565f> in <module>
     18
     19 bb = Chinese()
---> 20 bb.walk()

<ipython-input-22-63682224565f> in walk(self)
     12     def walk(self):      # 在子类中定义其自身的方法
     13         print(Chinese._name1, "is walking....")
---> 14         print(Chinese.__name2, "is walking....")
     15
     16 aa = Person()

AttributeError: type object 'Chinese' has no attribute '_Chinese__name2'
```

```
%%writefile test1.py
class Person(object):    # 定义一个父类

    _name1 = 'Tom'
    __name2 = 'Jerry'

    def talk(self):    # 父类中的方法
        print(Person._name1, "is talking....")
        print(Person.__name2, "is talking....")

    def __walk(self):
        print(Person._name1, "is walking....")
        print(Person.__name2, "is walking....")

    def _drink__(self):    # 父类中的方法
        print(Person._name1, "is drinking....")
```

```
Overwriting test1.py
```

```
import test1

a = test1.Person()
a.talk()
a._drink__()
```

属性的访问控制

get

```python
class Person(object):
    def __init__(self):
        self.__age = 18 # 定义一个私有化属性，属性名字前加连个 __ 下滑线

    @property # 使用装饰器对age进行装饰，提供一个getter方法
    def age(self): # 访问私有实例属性
        return self.__age

    @age.setter # 使用装饰器进行装饰，提供一个setter方法
    def age(self, age): # 修改私有实例属性
        if age < 0:
            print('年龄不能小于0')
        else:
            self.__age = age
xiaoming = Person()
xiaoming.age = 15
print(xiaoming.age)

# 来源：https://blog.csdn.net/MarconiYe/article/details/120165472
```

只读属性设置：

```
class Person(object):
    def __init__(self):
        self.__age = 18 # 定义一个私有化属性，属性名字前加连个 __ 下滑线

    @property # 使用装饰器对age进行装饰，提供一个getter方法
    def age(self): # 访问私有实例属性
        return self.__age

xiaoming = Person()
print(xiaoming.age)
xiaoming.age = 15
```

```
18

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-25-99f8120768d8> in <module>
      9 xiaoming = Person()
     10 print(xiaoming.age)
---> 11 xiaoming.age = 15

AttributeError: can't set attribute
```

```
%%writefile test1.py
class Person(object):    # 定义一个父类

    _name1 = 'Tom'
    __name2 = 'Jerry'

    def talk(self):    # 父类中的方法
        print(Person._name1, "is talking....")
        print(Person.__name2, "is talking....")

    def __walk(self):
        print(Person._name1, "is walking....")
        print(Person.__name2, "is walking....")

    def _drink__(self):    # 父类中的方法
        print(Person._name1, "is drinking....")
```

```
Overwriting test1.py
```

```
import test1

a = test1.Person()
a.talk()
a._drink__()
a.__walk()
```

```
Tom is talking....
Jerry is talking....
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[35], line 5
      3 a = test1.Person()
      4 a.talk()
----> 5 a._drink__()
```

Python的类型提示（by 王瑞环 助教）

- 在Python 3.5, 加入了“**函数参数与返回值类型提示**”的功能; 在Python 3.6, 加入了“**变量声明时类型注解**”的功能.两者的使用格式如下

```
def func(
        var_1: type_1 [= default_val_1],
        var_2: type_2 [= default_val_2],
        ...,
        var_n: type_n [= default_val_n]
    ) -> result_type:
    pass
```

```
variable: type_ = ...
```

- 在变量和类型定义的后面加入类型提示

```
def sumint(a:int, b:int=3) -> int:
    return a + b

c:int = sumint(3, 5)
print(c, type(c))
```

```
8 <class 'int'>
```

```
d:int = sumint('3', '5')
print(d, type(d))
```

```
35 <class 'str'>
```

The Python runtime does not enforce function and variable type annotations. They can be used by third party tools such as type checkers, IDEs, linters, etc.

```
# %pip install mypy
```

静态类型检查

```
%%writefile test_mypy.py
def add(x: int, y: int) -> int:
    result = x + y
    print(result)
    return result

add("3", "4")
```

```
Overwriting test_mypy.py
```

```
!mypy test_mypy.py
```

```
test_mypy.py:6: error: Argument 1 to "add" has incompatible type "str"; expect
ed "int"  [arg-type]
test_mypy.py:6: error: Argument 2 to "add" has incompatible type "str"; expect
ed "int"  [arg-type]
Found 2 errors in 1 file (checked 1 source file)
```

Python设计模式

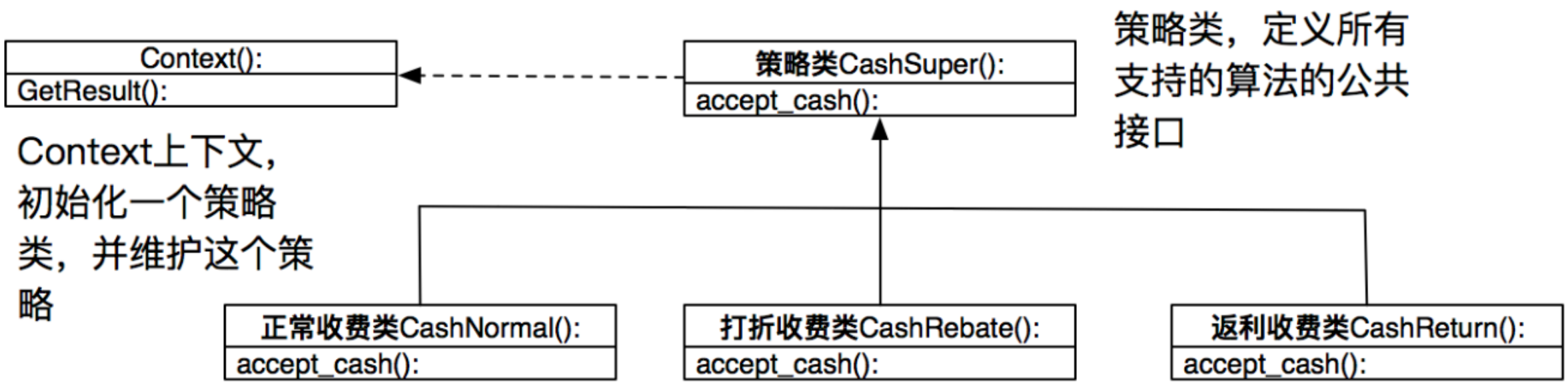
- 设计模式是面型对象方法里的一种解决方案的抽象
- 目的是把一些常见的应用抽象为一种类设计模式，在具体实现中只要套用或稍作修改，就能完成逻辑清晰的类实现方案
- 通过对设计模式的学习也可以达到对类体系的深入理解

python设计模式之：策略模式

经典的策略模式，由三部分组成：

- Context：上下文环境类
- Stragety：策略基类
- ConcreteStragety：具体策略

请阅读以下实现了python策略模式的代码，从而对python类、对象等概念有进一步的了解



策略类定义

```python
# 现金收费抽象类
class CashSuper(object):
    def accept_cash(self, money):
        pass

class CashNormal(CashSuper):
    """策略1: 正常收费子类"""
    def accept_cash(self, money):
        return money

class CashRebate(CashSuper):
    """策略2:打折收费子类"""
    def __init__(self, discount=1):
        self.discount = discount

    def accept_cash(self, money):
        return float(money) * float(self.discount)

class CashReturn(CashSuper):
    """策略3 返利收费子类"""
    def __init__(self, money_condition=0, money_return=0):
        self.money_condition = money_condition
        self.money_return = money_return

    def accept_cash(self, money):
        money = float(money)
        if money >= self.money_condition:
            return money - (money / self.money_condition) * self.money_return
        return money
```

```python
# 具体策略类
class Context(object):
    def __init__(self, cash_super):
        self.cash_super = cash_super

    def GetResult(self, money):
        return self.cash_super.accept_cash(money)


if __name__ == '__main__':
    money = input("原价: ")
    strategy = {}
    strategy[1] = Context(CashNormal())
    strategy[2] = Context(CashRebate(0.8))
    strategy[3] = Context(CashReturn(100, 10))
    mode = int(input("选择折扣方式: 1) 原价 2) 8折 3) 满100减10: "))
    if mode in strategy:
        cash_super = strategy[mode]
    else:
        print("不存在的折扣方式")
        cash_super = strategy[1]
    print("需要支付: ", cash_super.GetResult(money))
```

策略上下文环境类

```
原价: 800
选择折扣方式: 1) 原价 2) 8折 3) 满100减10: 3
需要支付:  720.0
```

单例模式

- 保证一个类仅有一个实例，并提供一个访问它的全局访问点。
- 避免一个全局使用的类重复创建与销毁。

单例模式 — 所有生成实例都指向同一个对象

```
class Singleton(object):

    attr = None                          # 类属性

    def __init__(self):
        print("Do something.")

    def __new__(cls, *args, **kwargs):           # 重载__new__()方法
        if not cls.attr:
            cls.attr = super(Singleton,cls).__new__(cls)

        return cls.attr

obj1 = Singleton()
obj2 = Singleton()
print(obj1, obj2)
```

调用 __new__()方法创建实例对象

```
Do something.
Do something.
<__main__.Singleton object at 0x00000215E55BC308> <__main__.Singleton object at 0x00000215E55BC308>
```

单例模式 — 所有生成实例都指向同一个对象

```
class Singleton(object):
    def __init__(self):
        print("Do something new.")

    def __new__(cls, *args, **kwargs):
        if not hasattr(Singleton, "_instance"):
            Singleton._instance = object.__new__(cls)

        return Singleton._instance

obj1 = Singleton()
obj2 = Singleton()
print(obj1, obj2)
```

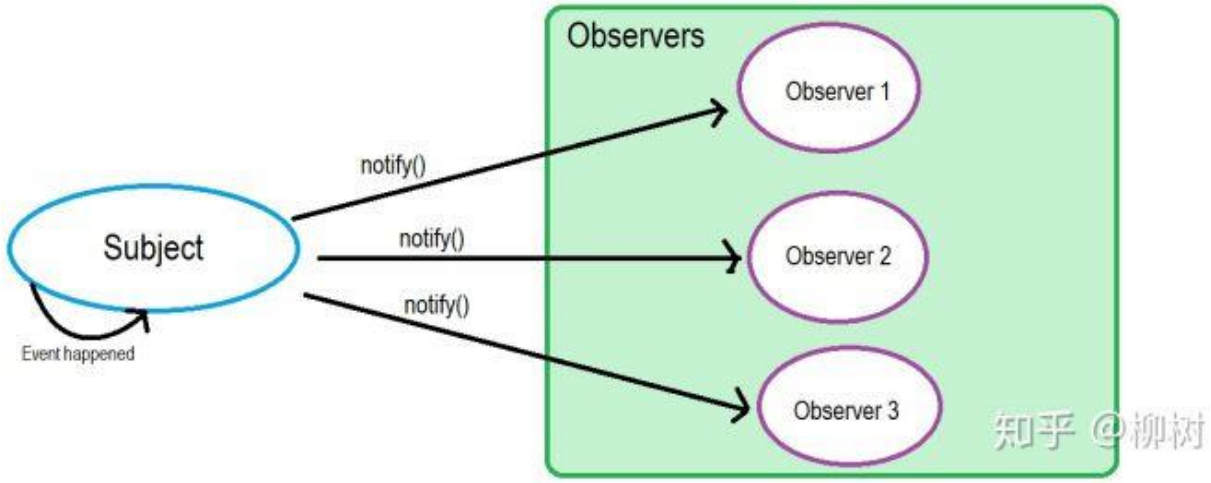
在cls空间动态生成一个内部属性
（另一种实现单例模式的方案）

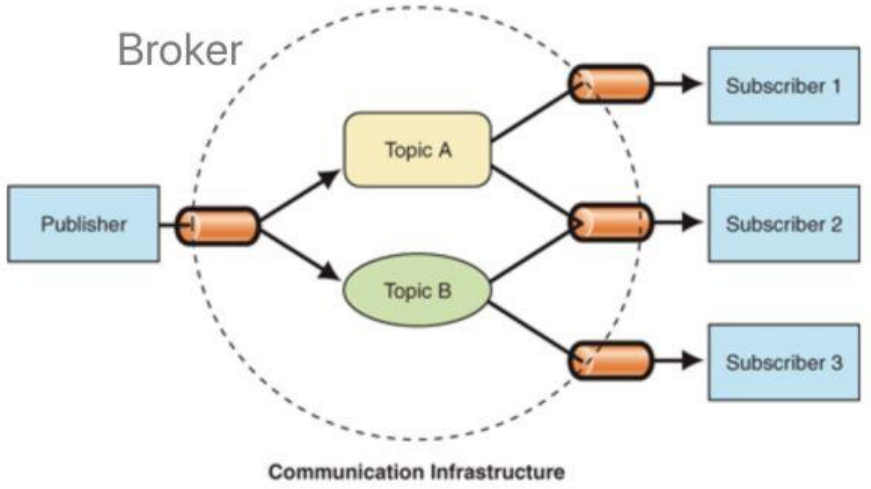
```
Do something new.
Do something new.
<__main__.Singleton object at 0x00000215E55B2A88> <__main__.Singleton object at 0x00000215E55B2A88>
```

```python
instances = {} # 一个全局的 类-实例 对的记忆buffer

def singleton(cls):    # 接受一个类
    def get_instance(*args, **kw):
        cls_name = cls.__name__      # 获得当前类名称
        print('已经创建过了')
        if not cls_name in instances: # 过去没有创建实例
            print('第一次创建')
            instance = cls(*args, **kw)  # 创建实例
            instances[cls_name] = instance #加入记忆buffer
        return instances[cls_name] # 返回实例引用
    return get_instance  # 返回装饰器函数

@singleton
class User:
    _instance = None    //?

    def __init__(self, name):
        print('===== 3 ====')
        self.name = name

User("xiao wang")
```

观察者模式（发布-订阅模式？）

- 多使用一种“注册—通知—撤销注册”的形式
- Observer）将自己注册到被观察对象（Subject）中，被观察对象将观察者存放在一个容器（Container）里
- 被观察对象发生了某种变化，从容器中得到所有注册过的观察者，将变化通知观察者





Pub-Sub Pattern (image credit: MSDN blog)

```python
class Observer:
    """
    观察者
    """
    def __init__(self, name):
        self.name = name

    def update(self, msg):
        print(self.name + "收到信息：" + msg)


class Subject:
    """ 某主题 """
    def __init__(self):
        self.observers = []

    def add_observers(self, observer):
        self.observers.append(observer)    # 这里利用了list的appen
        return self

     def remove_observer(self, observer):
        self.observers.remove(observer)
        return self

    def notify(self, msg):
        for observer in self.observers:
            observer.update(msg)
```

```python
xiaoming = Observer("xiaoming")
lihua = Observer("lihua")

rain = Subject()  # 生成主题。可以有主题词？

# 添加订阅
rain.add_observers(xiaoming)
rain.add_observers(lihua)

rain.notify("下雨了！")

# 取消订阅
rain.remove_observer(lihua)  # 可以主动订阅？ 条件约束订阅？
```

```
xiaoming收到信息：下雨了！
lihua收到信息：下雨了！
```

工厂模式简介：用来实现参数化定制类

- 工厂模式的本质上是用类定制类，然后到具体实例
- 工厂模式的基础是共性抽象，是把相关类的共性和个性化定制相融合的解决方案

工厂类的示例:

```
class StandardFactory(object):

    @staticmethod                          # 静态方法
    def get_factory(factory):    # 实际可以传入更多的参数
        '''根据参数找到对实际操作的工厂'''
        if factory == 'cat':
            return CatFactory()   # 这里如果要带参数，就会用到类属性，类方法
        elif factory == 'dog':
            return DogFactory()
        raise TypeError('Unknown Factory.')

class DogFactory(object):
    def get_pet(self): # 这里还可以带参数，甚至组合其他类，来定义不同类的dog
        return Dog(); # 返回一个dog类的实例

class CatFactory(object):
    def get_pet(self):
        return Cat();
```

工厂类的示例（抽象类）：

```
class Pet(abc.ABC):                 # 抽象类可以通过MyIterable方法来查询所有的派生子类
    @abc.abstractmethod             # 强制子类必须实现此方法
    def eat(self):
        pass

    def jump(self):                 # 不能创建实例，但可以被继承
        print("jump...")


# Dog类的具体实现
class Dog(Pet):
    def eat(self):    # 必须实现抽象类Pet中规定的方法
        return 'Dog eat...'

class Cat(Pet):
    def eat(self):
        return 'Cat eat...'
```

工厂类的示例（实际使用）：

```
if __name__ =="__main__":       # 如果被包含则 __name__ 会等于模块名，下面代码不会
    factory = StandardFactory.get_factory('cat')  # 配置抽象工厂参数，生成一个猫
    cat = factory.get_pet()                       # 生成一个猫实例
    print (cat.eat())                             # cat eat

    factory = StandardFactory.get_factory('dog')
    dog = factory.get_pet() ##这里工厂的操作与上面的生成cat是完全一样的，但结果
    print (dog.eat())                             # dog eat
    dog.jump()                                    # 继承自抽象类的jump
    cat.jump()
    #Pet().jump() TypeError: Can't instantiate abstract class Pet with abstract
```

```
Cat eat...
Dog eat...
jump...
jump...
```

总结一下Python的类实现方案

- 全面实现了面向对象的解决方案
- 提供了更高的灵活性
- 兼容了函数式编程模式
- 为参数化软件架构编程提供了好的支持

Python类编程架构与函数编程架构总结

- ‘类’定义了一个私有的数据和计算子环境。类又是可以通过继承、组合和派生（函数或工厂类）被生成出来的。类也可以是函数。
- 设计模式只是一些常见解决方案的经验模式，课程上主要用于对类编程技术进行学习，实现一些较复杂的类体系架构下的编程实现

正则表达式与模式匹配

- 可以用来判断某个字符串是否符合某种模式，比如判断某个字符串是不是邮箱地址，网址，电话号码，身份证号
- 可以用来在文本中寻找并抽取符合某种模式的字符串，比如电子邮件地址，网址，身份证号
- 正则表达式规定了一种模式，是一个字符串，如“abc”,“a.?p.*c”

正则表达式字符匹配

字符			
一般字符	代表自身。	abc	abc
.	匹配任意除换行符\n外的字符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。	a\.c a\\c	a.c a\c
[…]	字符集。对应的位置可以是字符集中任意字符。	a[bcd]e a[b-d]e	abe ace ade
[^abc]	第一个字符是^表示取反。		匹配abc之外的任意一个字符

预定义字符，可以在字符集中使用

字符			
\d	数字[0-9]	a\dc	a2c
\D	非数字[^\d]	a\Dc	abc
\s	空白字符[<空格>\n\t\r]	a\sc	a c
\S	非空白字符	a\Sc	abc
\w	单词字符[A-Za-z0-9]	a\wc	abc
\W	非单词字符[^\w]	a\Wc	a c

数量词字符

字符			
*	匹配**前**一个字符0或无数次	abc*	ab abcccc
+	匹配前一个字符1次或无数次	abc+	abc abcccc
?	匹配前一个字符0次或1次	abc?	ab abc
{m}	匹配前一个字符m次	ab{3}c	abbbc
{m, n}	匹配前一个字符m至n次，可省略	ab{2, 3}c	abbc abbbc
[\dabc]+	长度不为0的由数字或a、b、c构成的字符串		123abc 123abc123abc

正则表达式示例

`[1-9]\d*`	正整数
`-[1-9]\d*`	负整数
`-?[1-9]\d*|0`	整　数
`-?([1-9]\d*.\d*|0.\d*[1-9]\d*|0?.0+|0)`	小　数

注：｜表示“或”，采用短路匹配方案

Python与正则表达式——import re

re.match(pattern, string, flags=0)

从字符串的**起始位置**一个模式pattern

flags是标志位，用于控制模式串的匹配方式，标记是否区分大小写，多行匹配等，

re.I　使匹配不区分大小写。

re.M　跨多行匹配，影响^和$。

re.S　使.匹配所有字符。

```python
# re.match(pattern, string, flags)
import re

m1 = re.match("ab*c", "abbcde")
print('m1: ', m1 != None)

m2 = re.match("ab+c", "ac")
print('m2: ', m2 != None)

m3 = re.match("a.?bc.*", "abcd")
print('m3: ', m3 != None)

m4 = re.match("a.?bc.*", "aBcd")
print('m4: ', m4 != None)

m5 = re.match("a.?bc.*", "aBcd", re.I) # 修饰符re.I不区分大小写
print('m5: ', m5 != None)
```

```
m1:  True
m2:  False
m3:  True
m4:  False
m5:  True
```

re.search(pattern, string[, flags])

查找字符串中可以匹配成功的子串

匹配成功则返回一个匹配对象，失败返回None

```
# re.search(pattern, string[, flags])
import re
print('1: ', re.search("a.+bc*$", "mnadewbc")) # '$'表示处于字符串的结尾
print('2: ', re.search("a.+bc*$", "mnadewbcd"))
print('3: ', re.search("^a.+bc*", "mnadewbcd")) # '^'表示与字符开始处进行匹配
print('4: ', re.search("^a.+bc*", "adewbcd"))
```

```
1:  <re.Match object; span=(2, 8), match='adewbc'>
2:  None
3:  None
4:  <re.Match object; span=(0, 6), match='adewbc'>
```

re.findall(pattern, string[, flags])

查找字符串中所有和模式匹配的子串放入列表，一个子串都找不到则返回[]

```
# re.findall(pattern, string[, flags])
import re
sentence = '2020 is a leap year with 366 days.'
print('number: ',re.findall('\d+', sentence))
print('word: ', re.findall('[A-Za-z]+', sentence))
```

```
number:  ['2020', '366']
word:  ['is', 'a', 'leap', 'year', 'with', 'days']
```

匹配边界控制符

\A 与字符串开始的位置匹配，不消耗任何字符。

\Z 与字符串结束的位置匹配，不消耗任何字符。

^ 与字符串开始的位置匹配，不消耗任何字符。在多行模式中，匹配每一行开头。

$ 与字符串结束的位置匹配，不消耗任何字符。在多行模式中，匹配每一行末尾。

\b 匹配一个单词的开始处和结束处，不消耗任何字符。

\B 和\b相反，不允许是单词的开始处和结束处。

```python
import re
test1 = "\Ahow are"
print('test1-1: ', re.search(test1, "hhow are you"))
print('test1-2: ', re.search(test1, "how are you").group())
test2 = "are you\Z"
print('test2-1: ', re.search(test2, "how are you?"))
print('test2-2: ', re.search(test2, "how are you").group())
test3 = "^how are"
print('test3-1: ', re.findall(test3, "how are you\nhow are you", re.M))
test4 = "are you$"
print('test4-1: ', re.search(test4, "how are you\nhow are you?", re.M).group())
test5 = r"\bA.*B\b C"
print('test5-1: ', re.search(test5, "Abb$BD CD"))
print('test5-2: ', re.search(test5, "test Abb$B CD").group())
test6 = r"\BA.*B\B\w C"
print('test6-1: ', re.search(test6, "test Aab$B CD"))
print('test6-2: ', re.search(test6, "testAab$BE CD").group())
```

\A 与字符串开始的位置匹配

\Z 与字符串结束的位置匹配

```
test1-1:  None
test1-2:  how are
test2-1:  None
test2-2:  are you
test3-1:  ['how are', 'how are']
test4-1:  are you
test5-1:  None
test5-2:  Abb$B C
test6-1:  None
test6-2:  Aab$BE C
```

匹配边界控制示例

正则表达式中的“分组”

括号中的表达式是一个分组，多个分组按左括号从左到右从1开始编号。

```
# group
import re
pattern = "(((ab*)c)d)e"
r = re.match(pattern, "abbcdefg")
print('group:   ', r.groups())
print('index:   ', r.lastindex)
print('group 0:', r.group(0))
print('group 1:', r.group(1))
print('group 2:', r.group(2))
print('group 3:', r.group(3))
```

```
group:    ('abbcd', 'abbc', 'abb')
index:    1
group 0: abbcde
group 1: abbcd
group 2: abbc
group 3: abb
```

re.finditer(pattern, string[, flags])

在字符串中找到正则表达式所匹配的所有子串，并作为一个迭代器返回

```
import re
s = '123[45]67<890>a[bc]ba<098>'
m = '\[(\d+)\]|<(\d+)>'
for x in re.finditer(m,s):
    print(x.group())
```

```
[45]
<890>
<098>
```

```
import re
p = r"(((ab*)+c|12+3)d)e"
for x in re.finditer(p,'ababcdef12def1223def'):
    print (x.group())    # group 等价于group(0)
    print (x.groups())   # groups返回一个元祖，元素依次是1-3号分组所匹配的字符串
```

```
ababcde
('ababcd', 'ababc', 'ab')
1223de
('1223d', '1223', None)
```

re.sub(pattern, repl, string)

```
import re
phone = '010-1234-5678'
num = re.sub(r'\D', "", phone)
print("Phone Number : ", num)
```

```
Phone Number :  01012345678
```

用于替换匹配的子串，
repl可以是替换串，
也可以是函数。

```
import urllib.request
url = "https://its.pku.edu.cn"
file = urllib.request.urlopen(url)
info = file.info()
print(info)
```

```
Date: Sun, 01 Mar 2020 06:53:12 GMT
Server: Apache
Set-Cookie: JSESSIONID=57442C88BECAFCBF64BDD23B2C7E1F20; Path=/; Secure; HttpOnly
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html;charset=utf-8
```

```
new_info = re.sub('(.+): (.+)',
           lambda x: "'{}': '{}'.".format(x.group(1), x.group(2)),
           str(info))
print(new_info)
```

```
'Date': 'Sun, 01 Mar 2020 06:53:12 GMT'.
'Server': 'Apache'.
'Set-Cookie': 'JSESSIONID=57442C88BECAFCBF64BDD23B2C7E1F20; Path=/; Secure; HttpOnly'.
'Connection': 'close'.
'Transfer-Encoding': 'chunked'.
'Content-Type': 'text/html;charset=utf-8'.
```

数量词的贪婪模式与非贪婪模式

```
import re
# 量词+,*,?,{m,n}默认匹配尽可能长的字符串 贪婪模式
pattern = "<t>.*</t>"
string = "<t>abcd</t><t>abcde</t>"
test1 = re.match(pattern, string)
print('test1: ', test1.group())

# 在量词+,*,?,{m,n}后面加'?'匹配尽可能短的字符串 非贪婪模式
pattern1 = "<t>.*?</t>"
test2 = re.match(pattern1, string)
print('test2: ', test2.group())
```

```
test1:  <t>abcd</t><t>abcde</t>
test2:  <t>abcd</t>
```

计算机网络

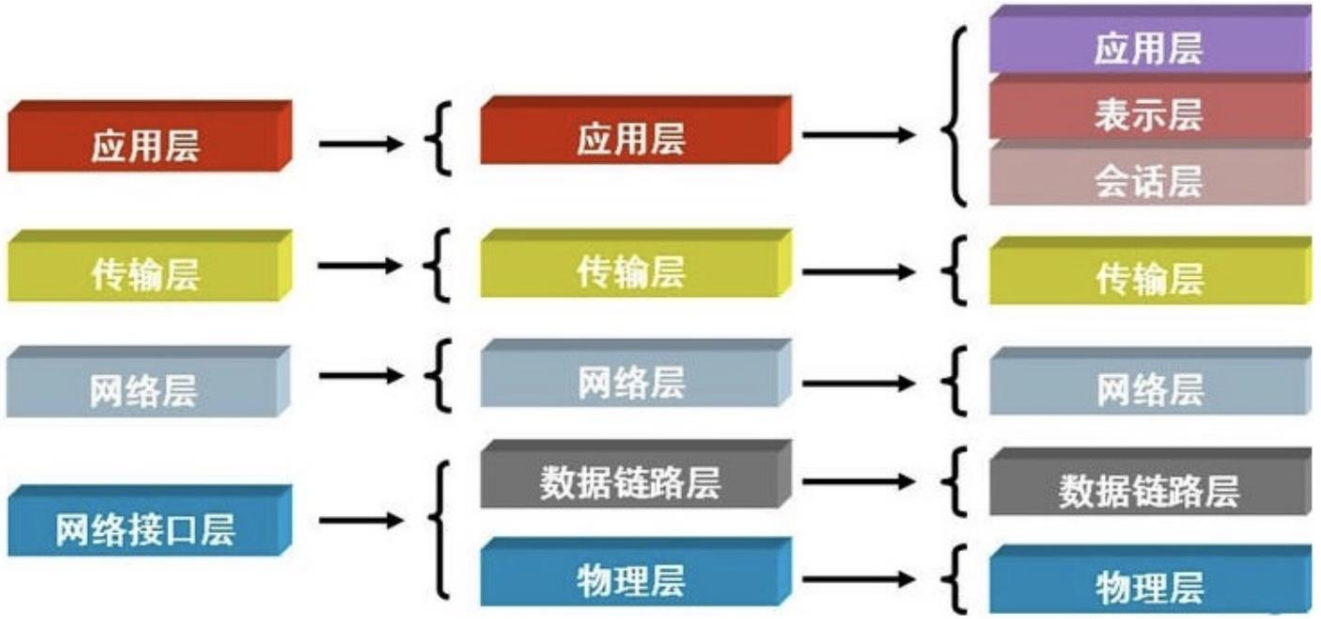
- TCP/IP通讯协议与C/S

什么是计算机网络?

- 计算机网络：用通信设备将计算机连接起来，在计算机之间传输数据（信息）的系统。
- 连网的计算机根据其提供的功能将之区分为客户机或服务器（C/S）
- 通信协议：计算机之间以及计算机与设备之间进行数据交换而遵守的规则、标准或约定
  - 典型的协议：TCP/IP（在互联网上采用），IEEE802.3以太网协议（局域网），IEEE902.11（无线局域网，WIFI）

开放系统互连模型OSI

- 物理层：利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。
- 数据链路层：将比特组成数据帧，临近网络部件间的数据传送
- 网络层：源到目的之间基于分组的数据交换
- 传输层：提供端对端的透明数据传输服务
- 会话层：不同主机的进程间会话的组织和同步
- 表示层：为上层用户提供共同需要的数据或信息语法表示及转换
- 应用层：为用户提供服务，提供网络应用



客户端-服务器模型

- 一个应用由一个服务器进程和一个或多个客户端进程组成。
- 服务器管理某种资源，并且通过操作这种资源来为它的客户端提供某种服务。

1. 客户端向服务器发送一个请求。

   Web浏览器需要文件时，发送请求给Web服务器。

2. 服务器收到请求后，解释它，并以适当的方式操作资源。

   Web服务器收到浏览器发出的请求后，读一个磁盘文件。

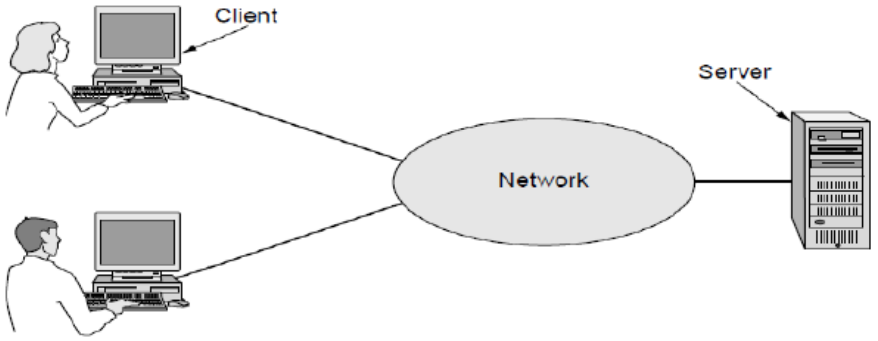
3. 服务器给客户端发送一个响应，并等待下一个请求。

   Web服务器将文件发送回客户端。

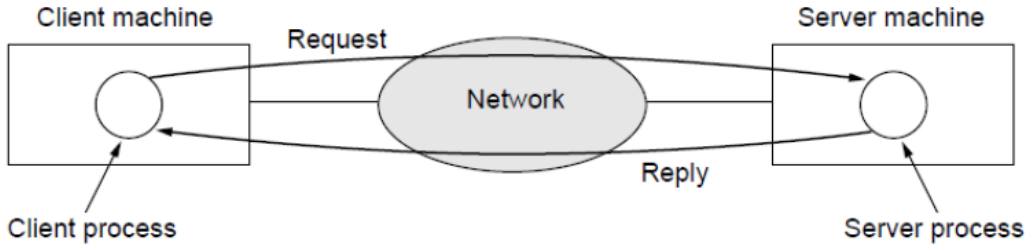
4. 客户端收到响应并处理它。

   Web浏览器收到来自服务器的一页后，在屏幕上显示此页。

有两个客户机一个服务器的网络



在客户机服务器模式中包含请求和响应



网络通讯实现的基本方案：

- 服务器监听特定的端口：如8080
- 得到请求后建立数据链路（远程I/O strem）
- 启动阻塞式通讯（TCP）

IP+port通讯

信息：需要寄的快递

IP：小区

Port：门牌号,共有65536个端口

Socket：快递地址（小区+门牌号）

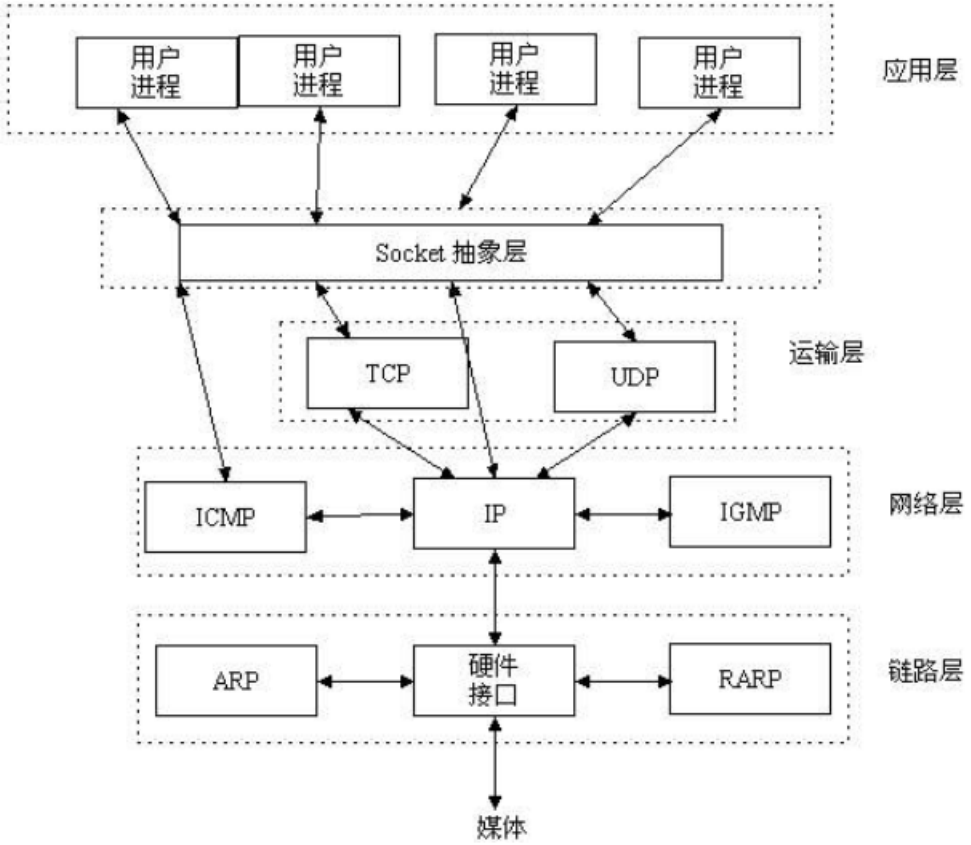
TCP，UCP等协议：快递公司

利用socket发送消息：把快递（消息）放到门口（socket），由快递公司（TCP等协议）负责送到对应的地址（对方socket）

传输层协议

- TCP：传输控制协议，面向连接、可靠。适用于要求可靠传输的应用。
  面向连接：发送数据之前必须在两端建立连接。
  仅支持单播传输：只能进行点对点数据传输。
  面向字节流：在不保留报文边界的情况下以字节流的方式进行传输。
  可靠：对每个包赋予序号，来判断是否出现丢包、误码。

- UDP：用户数据报协议，面向非连接、不可靠。适用于实时应用。
  面向非连接：发送数据不需要建立连接。
  支持单播、多播、广播
  面向报文：对应用层的报文添加首部后直接向下层交付。
  不可靠：没有拥塞控制，不会调整发送速率。



Socket是传输层和应用层之间的软件抽象层，是一组接口。

对于用户来说，socket把复杂的TCP/IP协议族隐藏在接口后，只需要遵循socket的规范，就能得到遵循TCP/UDP标准的程序。

Socket通信

套接字socket：网络中不同主机上的应用进程之间进行双向通信的端点。

每台主机有一个唯一的主机地址标识（IP），同时主机内还有标识服务的序号id，称作端口（port）。

socket绑定了相应的IP和port，可以用（IP：port）的形式表示一个socket地址。

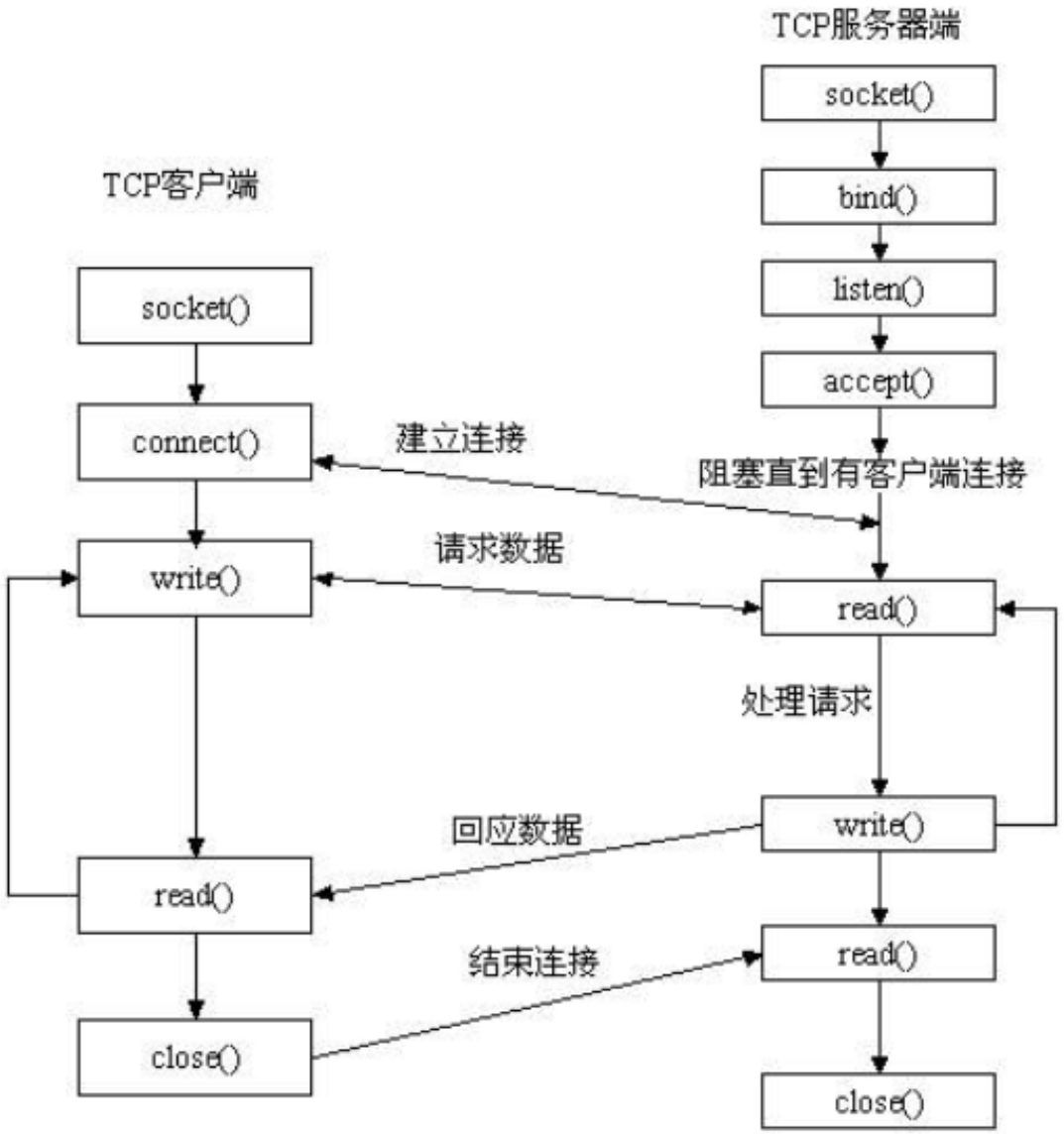
当客户端发起一个连接请求时，客户端socket地址中的端口由系统自动分配，服务器端套接字地址中的端口通常是某个和服务相对应的知名端口。（例如Web服务器常使用端口80，电子邮件服务器使用端口25）

一个连接由它两端的socket地址唯一确定：

(ClientIP : ClientPort, ServerIP : ServerPort)

创建套接字: `socket.socket(family, type)`
参数说明:
family：套接字家族，可以使AF_UNIX或者AF_INET，一般是AF_INET。
type：套接字类型，根据是面向连接的还是非连接分为SOCK_STREAM或SOCK_DGRAM，也就是TCP和UDP的区别，一般是SOCK_STREAM。

socket类型	**描述**
socket.AF_UNIX	只能够用于单一的Unix系统进程间通信
socket.AF_INET	IPv4
socket.AF_INET6	IPv6
socket.SOCK_STREAM	流式socket , for TCP
socket.SOCK_DGRAM	数据报式socket , for UDP
socket.SOCK_RAW	原始套接字，普通的套接字无法处理ICMP、IGMP等网络报文，而SOCK_RAW可以；其次，SOCK_RAW也可以处理特殊的IPv4报文；此外，利用原始套接字，可以通过IP_HDRINCL套接字选项由用户构造IP头。
socket.SOCK_SEQPACKET	可靠的连续数据包服务
创建TCP Socket：	s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
创建UDP Socket：	s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)



服务器端：
初始化socket，与IP端口绑定，对IP端口进行监听，调用accept()阻塞，等待客户端连接。

客户端：
初始化socket，连接服务器。

连接成功后客户端发送数据请求，服务器端接收并处理请求、回应数据，客户端读取数据。

最后关闭连接，一次交互结束。

服务器端方法	
s.bind()	绑定地址（host,port）到套接字，在AF_INET下,以元组（host,port）的形式表示地址。
s.listen(backlog)	开始监听。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。
s.accept()	被动接受客户端连接,(阻塞式)等待连接的到来，并返回（conn,address）二元元组,其中conn是一个通信对象，可以用来接收和发送数据。address是连接客户端的地址。

客户端方法	
s.connect(address)	客户端向服务端发起连接。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。
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

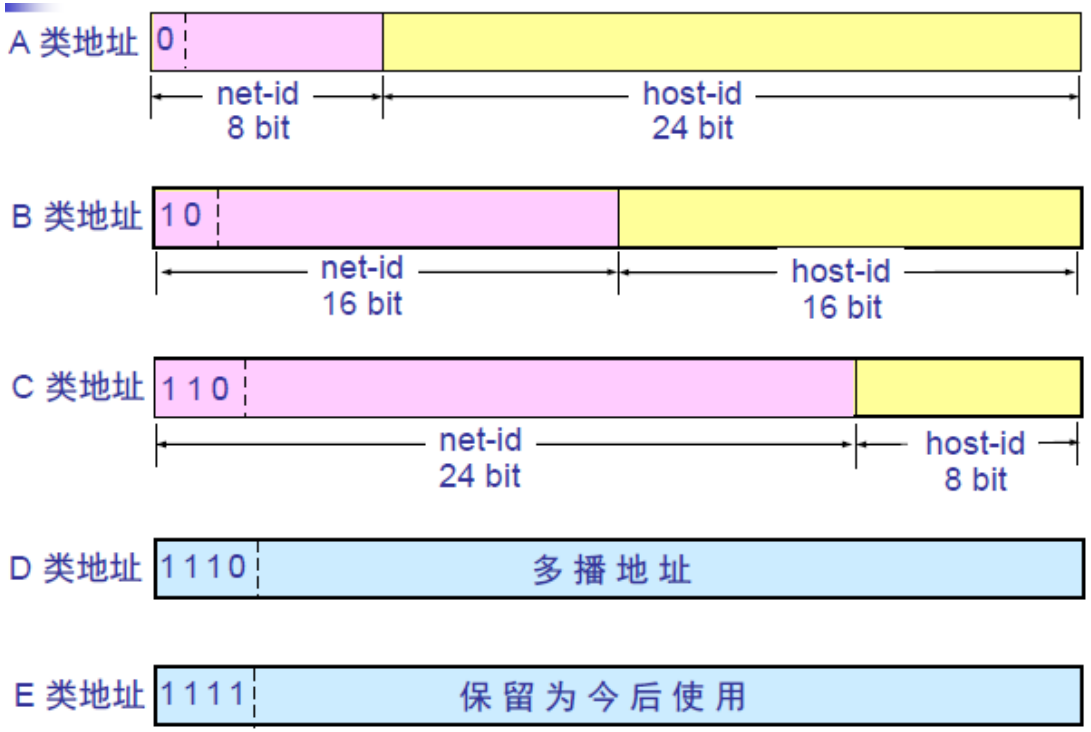
s.recv(bufsize)	接收数据，数据以bytes类型返回，bufsize指定要接收的最大数据量。
s.send()	发送数据。返回值是要发送的字节数量。
s.sendall()	完整发送数据。将数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。
s.recvform() s.recvfrom()	接收UDP数据，与recv()类似，但返回值是（data,address）。其中data是包含接收的数据，address是发送数据的套接字地址。
s.sendto(data,address)	发送UDP数据，将数据data发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地址。返回值是发送的字节数。
s.close()	关闭套接字，必须执行。
s.getpeername()	返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。
s.getsockname()	返回套接字自己的地址。通常是一个元组(ipaddr,port)
s.setsockopt(level,optname,value)	设置给定套接字选项的值。
s.getsockopt(level,optname[.buflen])	返回套接字选项的值。
s.settimeout(timeout)	设置套接字操作的超时期，timeout是一个浮点数，单位是秒。值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如connect()）

- IP地址：IPv4 – 32位，IPv6 – 128位
- IP地址分类：每个地址由两个固定长度的字段组成，网络号net-id标志主机所连接到的网络，主机号host-id标志该主机。

127.0.0.1和0.0.0.0的区别：

回环地址127.x.x.x：该范围内的任何地址都将环回到本地主机中，不会出现在任何网络中。主要用来做回环测试。

0.0.0.0：任何地址，包括了环回地址。不管主机有多少个网口，多少个IP，如果监听本机的0.0.0.0上的端口，就等于监听机器上的所有IP端口。数据报的目的地址只要是机器上的一个IP地址，就能被接受。



单线程服务端

```python
import socket
import time
# 定义服务器信息
print('初始化服务器主机信息')
port = 5002  #端口  0--1024 为系统保留
host = '0.0.0.0'
address = (host, port)
# 创建TCP服务socket对象
print("初始化服务器主机套接字对象......")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 关掉连接释放掉相应的端口
# server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# 绑定主机信息
print('绑定的主机信息......')
server.bind(address)
# 启动服务器 一个只能接受一个客户端请求，可以有1个请求排队
print("开始启动服务器......")
server.listen(5)
#等待连接
while True:
    # 等待来自客户端的连接
    print('等待客户端连接')
    conn, addr = server.accept()  # 等电话
    print('连接的客服端套接字对象为：{}\n客服端的IP地址（拨进电话号码）：{}'.format(conn, addr))
    #发送给客户端的数据
    conn.send("欢迎访问服务器".encode('utf-8'))
    time.sleep(100)
    conn.close()
```

```python
#-*- coding: utf-8 -*-
import socket  # 导入 socket 模块

port = 5002
hostname = '127.0.0.1'

client = socket.socket()  # 创建 socket 对象
client.connect((hostname, port))
data = client.recv(100).decode('utf-8')
print(data)

client.close()
```

服务端输出:

```
初始化服务器主机信息
初始化服务器主机套接字对象......
绑定的主机信息......
开始启动服务器......
等待客户端连接
连接的客服端套接字对象为: <socket.socket fd=1092, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.
0.1', 5002), raddr=('127.0.0.1', 60984)>
客服端的IP地址（拨进电话号码）: ('127.0.0.1', 60984)
```

客户端输出:

```
$ python client.py
欢迎访问服务器
```

CS架构与BS架构：以买火车票为例

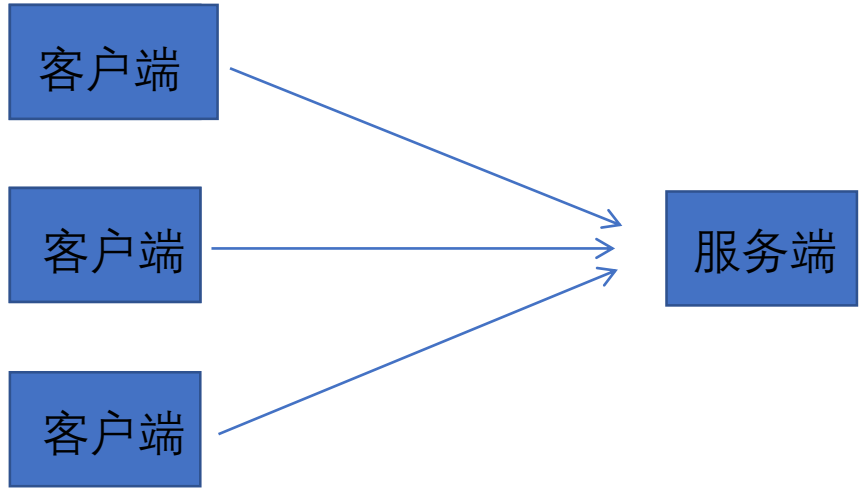
12306 APP买票是C/S服务模式：

客户端：

发出查询请求，建立链接。如果有则购买一张票

服务端：

维护余票情况，如果有余票则卖票给客户端，余票数量减一；没有则返回购买失败



用12306网页购票是B/S模式：

不保持链接。通过协议头设置一些参数。通过客户端发送cookie来识别特定用户。

界面交互事务逻辑在前端实现，主要事务逻辑在服务器端实现，常见可以和数据库端形成三层结构。

建立在广域网之上，只要有网络、浏览器，可以随时随地进行业务处理。

单进程服务端模拟购票

```python
# -*- coding: utf-8 -*-
import socket
import time

port = 5002
host = '0.0.0.0'

ticket_num = 2
def buy_ticket(conn):
    if_bought = 0
    global ticket_num
    if ticket_num > 0:
        ticket_num  -= 1
        if_bought = 1
    # 模拟信号传输时间
    time.sleep(5)
    conn.send((str(ticket_num) + str(if_bought)).encode('utf-8'))
    conn.close()
```

```python
# 定义服务器信息
print('初始化服务器主机信息')
address = (host, port)
# 创建TCP服务socket对象
print("初始化服务器主机套接字对象.....")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 关掉连接释放掉相应的端口
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# 绑定主机信息
print('绑定的主机信息.....')
server.bind(address)
# 启动服务器 一个只能接受一个客户端请求，可以有1个请求排队
print("开始启动服务器.....")
server.listen(5)
#等待连接
while True:
    # 等待来自客户端的连接
    print('等待客户端连接')
    conn, addr = server.accept()  # 等电话
    print('连接的客服端套接字对象为: {}\n客服端的IP地址（拨进电话号码）: {}'.format(conn
    buy_ticket(conn)
```

弊端：顺序，一个客户端堵塞会影响其余客户端

客户端

```python
#-*- coding: utf-8 -*-
import socket  # 导入 socket 模块

port = 5002
hostname = '127.0.0.1'

client = socket.socket()  # 创建 socket 对象
client.connect((hostname, port))
data = client.recv(100).decode('utf-8')
ticket_num,if_bought = int(data[:-1]),int(data[-1])
if not if_bought:
    print(f'现在还剩下{ticket_num}张票,客户端1没有买到票')
else:
    print(f'现在还剩下{ticket_num}张票,客户端1成功买到了一张票')
client.close()
```

B/S架构与HTTP协议

Python爬虫

1. 工具准备

➢第三方依赖库

- Requests: HTTP请求库
- BeautifulSoup: HTML解析库
- Pymongo: MongoDB的Python封装模块
- Selenium: Web自动化测试框架，用于模拟登陆和获取JS动态数据
- Pytessract: 一个OCR识别模块，用于验证码识别
- Pillow:　　Python图像处理模块

2. HTTP请求

➢HTTP的基本概念

通常HTTP消息包括客户端向服务器的请求消息和服务器向客户端的响应消息。
这两种类型的消息由一个起始行，一个或者多个头域，一个指示头域结束的空行和可选的消息体组成。

➢HTTP概览

Request URL: 表示请求的URL
Request Method: 表示请求的方法，还有HEAD, POST,DELETE,PUT等，其中GET和POST最常用。
Status Code: 显示HTTP请求和状态码，表示HTTP请求的状态。

- 1xx: 请求已被服务器接收，继续处理。
- 2xx: 请求已被服务器接收、理解并接受。
- 3xx: 需要后续操作才能完成这一请求。
- 4xx: 请求含有词法错误或者无法被执行。
- 5xx: 服务器在处理某个正确请求时发生错误。

2. HTTP请求示例：

```
import requests
url = 'https://its.pku.edu.cn'
data = requests.get(url)
print('status code: ', data.status_code)
```

```
status code:  200
```

```
print(list(dir(data)))
```

```
['__attrs__', '__bool__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__nonzero__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_content', '_content_consumed', '_next', 'apparent_encoding', 'close', 'connection', 'content', 'cookies', 'elapsed', 'encoding', 'headers', 'history', 'is_permanent_redirect', 'is_redirect', 'iter_content', 'iter_lines', 'json', 'links', 'next', 'ok', 'raise_for_status', 'raw', 'reason', 'request', 'status_code', 'text', 'url']
```

```
print(data.content)
```

```
b'<!-- ONLINE: 2017-12-30 -->\n<!doctype html>\n<html>\n<head>\n<meta charset="utf-8">\n<meta name="renderer" content="webkit">\n<meta http-equiv="X-UA-Compatible" content="IE=edge">\n<meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,user-scalable=no"/>\n<meta name="apple-mobile-web-app-status-bar-style" content="black" />\n<meta name="format-detection" content="telephone=no" />\n<title>\xe5\x8c\x97\xe4\xba\xac\xe5\xa4\xa7\xe5\xad\xa6\xe7\xbd\x91\xe7\xbb\x9c\xe6\x9c\x8d\xe5\x8a\xa1 - \xe9\xa6\x96\xe9\xa1\xb5</title>\n<link href="css/icon.css" rel="stylesheet" type="text/css">\n<link href="css/basez.css" rel="stylesheet" type="text/css">\n<link hr
```

3. 解析HTTP：使用BeautifulSoup解析处理

select函数：

1. 通过标签名查找 soup.select(‘title’)
2. 通过类名查找，前面加. soup.select(‘.class’)
3. 通过id查找，前面加# soup.select(‘#id’)
4. 组合查找 soup.select(‘title #id’) 中间有一个空格
5. 属性查找，属性需要使用中括号括起来。

3. 解析HTTP-使用BeautifulSoup解析处理

请关注，新学期有关通知信息汇总！（持续更新中）

2020/03/06

近期，新型冠状病毒疫情牵动人心，北京大学积极部署疫情防控，发布了《关于推迟2020年春季学期开学时间的通知》。新闻中心特整理关于学业、就业、出国等相关信息的通知，希望能够帮助大家更好地规划假期和春季学期，...

风雨同舟、守望相助、合作抗疫——境外合作高校与北京大学互致信函相互鼓励

2020/03/25

邱水平调研毕业生就业工作 要求多措并举强化关心关怀

2020/03/24

重大突破！中国南海可燃冰第二次试采成功：北大水合物中心卢海龙教授团队发挥重要作用

2020/03/27

信息管理系举办学习习近平总书记给北京大学援鄂医疗队全体“90后”党员回信精神主题党团日活动

2020/03/27

北京大学召开理工科研工作视频会

2020/03/26

```
<div class="headLine">…</div>
<!-- 头条结束 -->
<!-- 图片轮播开始 -->
<div class="Banner">…</div>
<!-- 图片轮播结束 -->
<div class="news-topic">
  <!-- 新闻纵横开始 -->
  <div class="news">
    <h2 class="listTitle">…</h2>
    <div style="display:none;">1</div>
    <div class="imgArticleList imgHover">…</div>
    <ul class="newsList">
      <li>
        <a href="xwzh/3832d39000ac4cc6b4a77484516ca27e.htm">风雨同舟、守望相助、合作抗疫—境外合作高校与北京大学互致信函相互鼓励</a>
        <p class="item-date-view">…</p>
      </li>
      <li>…</li>
      <li>…</li>
      <li>…</li>
      <li>…</li>
      <li>…</li>
    </ul>
  </div>
  <!-- 新闻纵横结束 -->
  <!-- 三个专题热点开始 -->
  <div class="topic">…</div>
  <!-- 三个专题热点结束 -->
  <!-- 三个专题网站开始 -->
```

4. 使用Cookie模拟登录

- Cookie保持登录机制

前一次登录时，服务器发送了包含登录凭据(用户名和密码的某种加密形式)的Cookie到用户的硬盘中。再次登录时，如果Cookie尚未到期，则浏览器会发送该cookie，服务器验证凭据，于是不必输入用户名和密码就可以让用户登录

图4-1. 微博登录实例

4. 使用Cookie模拟登录

- Cookie

  Cookie，可以简单认的为是在浏览器端记录包括登陆状态在内的各种属性值的容器名称，其实就是服务器为了保持浏览器与服务器之间连通状态，而在用户本地上创建的数据。只要用户再一次登陆，服务器会主动地寻找这些预存的数据，而无需再要求像第一次一样的操作。
- 使用Cookie的两种方式

  - 将Cookie写在header头部
  - 使用requests插入Cookie

    cookie = {“Cookie”: “xxxxxx”}

    html = requests.get(url, cookies=cookie)

6.使用Selenium

- Selenium简介
  - Selenium可以用来模拟浏览器的运行，直接运行在浏览器中，可以让浏览器自动加载页面，获取需要的数据，甚至页面截图，或者判断页面上某些动作是否发生。

```python
import requests
import time
import re
import json
import math

#目标网页地址，如：raw_url ='https://baijiahao.baidu.com/s?id=1746719319769828828&wfr=spider&for=pc'
raw_url = 'https://mbd.baidu.com/newspage/data/landingsuper?context=%7B%22nid%22%3A%22news_9513142528065658619%22%7D&n_type=-1&p_from=-1'

#设置协议头
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36 Edg/106
    "Connection": "close"
}

#提取前面的链接cookie值
res = requests.get(raw_url,headers = headers) # 按url获取网页
# print(list(dir(res))
# print(res.text)

#提取cookie为字典形式
cookie = res.cookies.get_dict()

#for i in cookie.items():
#    print(i)

print('cookie=',cookie.values)

obj = re.compile(r"\"tid\":\"(?P<thread_id>.*?)\",\"",re.S)  # match='"tid":"1034000052159141","', re.S 扩展到不含双引号的整个字符串
tidset = obj.finditer(res.text)  # find tid 正则表达式返回迭代器

#thread id 后面情节评论的url中要用
for i in tidset:
    print(i)
    thread_id = i.group('thread_id')
res.close() # 关闭链接
```