Day09-正则表示式vs爬虫程序

今日课程学习目标

```
熟悉正则表达式的基本语法
熟悉re正则模块中常用方法是使用
常握使用正则提取指定分组的数据
理解正则中的贪婪模式和非贪婪模式
熟悉 FastAPI 提取URL地址数据
熟悉爬虫的概念和作用
```

今日课程内容大纲

```
# 1. 正则表达式
    正则表达式的作用
    正则表达式的语法
        匹配单个字符
        匹配多个字符
        正则分组
    re正则模块的使用
    贪婪模式vs非贪婪模式
# 2. 爬虫程序
    FastAPI返回图片数据
    FastAPI提取URL地址数据
    爬虫的概念和作用
    requests简单使用
```

知识点1：正则表达式功能简介【熟悉】

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。

模式：**一种特定的字符串模式，这个模式是通过一些特殊的符号组成的。**

正则表达式的功能：

- 数据验证（表单验证、如手机、邮箱、IP地址）
- 数据检索（数据检索、数据抓取）
- 数据隐藏（`135****6235` 王先生）
- 数据过滤（论坛敏感关键词过滤）

> 正则表达式并不是Python所特有的，在Java、PHP、Go以及JavaScript等语言中都是支持正则表达式的。

知识点2：正则表式语法-匹配单个字符【熟悉】

正则表达式在线练习网址： https://tool.oschina.net/regex/

匹配单个字符：

正则语法	描述
.	匹配任意1个字符（除了\n）
[]	匹配[ ]中列举的字符
\d	匹配数字，即0-9
\D	匹配非数字，即不是数字
\s	匹配空白，即 空格，tab键
\S	匹配非空白
\w	匹配非特殊字符，即a-z、A-Z、0-9、_、汉字
\W	匹配特殊字符，即非字母、非数字、非汉字

练习：

- 在字符串 "abcd123" 中匹配 a：`a`
- 在字符串 "abcd123" 中匹配任意一个字符：`.`
- 在字符串 "abcd123" 中匹配 b 或 d：`[bd]`
- 在字符串 "abcd123" 中匹配数字: `\d`
- 在字符串 "abcd123" 中匹配非数字内容：`\D`
- 在字符串 "abcd 123" 中匹配空白字符串：`\s`
- 在字符串 "abcd 123" 中匹配非空白字符串：`\S`
- 在字符串 "abcd_123" 中匹配非特殊字符：`\w`
- 在字符串 "abcd&%123" 中匹配特殊字符：`\W`

知识点3：正则表达式-匹配多个字符【熟悉】

匹配多个字符：

正则语法	描述
*	匹配前一个字符出现0次或者无限次，即可有可无
+	匹配前一个字符出现1次或者无限次，即至少有1次
?	匹配前一个字符出现1次或者0次，即要么有1次，要么没有
{m}	匹配前一个字符出现m次
{m,n}	匹配前一个字符出现从m到n次

练习：

- 在字符串 "传智播客_python_666" 中匹配非数字内容：`\D*` 返回：`传智播客_python_`
- 在字符串 "传智播客_python_666" 中匹配数字：`\d+` 返回：`666`
- 在字符串 "传智播客_python_666" 中匹配y或py：`[p]?y` 返回：`py`
- 在字符串 "传智播客_python_666" 中匹配2个数字：`\d{2}` 返回：`66`
- 在字符串 "传智播客_python_666" 中匹配英文字母出现1-3次：`[a-z]{1,3}` 返回：`pyt` `hon`

知识点4：正则表达式-匹配开头结尾vs其他匹配【熟悉】

匹配开头和结尾：

正则语法	描述
^	匹配字符串开头
$	匹配字符串结尾

练习：

- 在字符串 "abc_python_666" 中匹配以a开头：`^a` 返回：`a`
- 在字符串 "abc_python_666" 中匹配以数字结尾：`\d$` 返回：`6`

其他匹配：

正则语法	描述
[^指定字符]	匹配除了指定字符以外的所有字符
|	匹配左右任意一个表达式

练习：

- 在字符串 "abc_python_666" 中匹配除了数字以外的字符：`[^\d]+` 返回：`abc_python_`
- 在字符串 "abc-python-666" 中匹配数字和特殊字符：`\d+|\W+` 返回：`-` `-` `666`

知识点5：re模块-match方法的使用【熟悉】

函数语法格式：

```
match函数：re.match(pattern, string, flags=0)
参数：
    pattern：匹配的正则表达式
    string：要匹配的字符串
    flags：标志位(暂时忽略，后面会介绍)

功能：尝试从字符串起始位置匹配一个正则表达式
    1）如果不能从起始位置匹配成功，则返回None；
    2）如果能从起始位置匹配成功，则返回一个匹配的对象
```

示例代码：

```
my_str = 'abc_123_DFG_456'

# 匹配字符串bc(注：从头开始)
res = re.match(r'bc', my_str)
print(type(res), res) # 注意：不能从起始位置匹配成功，则返回None


# 匹配字符串abc(注：从头开始)
res = re.match(r'abc', my_str)
print(type(res), res) # 注意：能从起始位置匹配成功，则返回一个匹配的对象，Match类实例对象

# Match对象.group()：获取正则表达式匹配到的内容
print(res.group())
```

知识点6：re模块-search方法的使用【熟悉】

函数语法格式：

```
search函数：re.search(pattern, string, flags=0)
参数：
    pattern：匹配的正则表达式
    string：要匹配的字符串
    flags：标志位(暂时忽略，后面会介绍)

功能：根据正则表达式扫描整个字符串，并返回第一个成功的匹配
    1）如果不能匹配成功，则返回None；
    2）如果能匹配成功，则返回一个匹配对象
```

示例代码：

```
my_str = 'abc_123_DFG_456'

# 匹配连续的3位数字
# \d：匹配一位数字
res = re.search(r'\d{3}', my_str)
print(type(res), res) # 注意：匹配成功，则返回一个匹配的对象，Match类实例对象

# 获取正则表达式匹配到的内容
print(res.group())
```

知识点7：re模块-findall方法的使用【熟悉】

函数语法格式：

```
findall函数：re.findall(pattern, string, flags=0)
参数：
    pattern：匹配的正则表达式
    string：要匹配的字符串
    flags：标志位(暂时忽略，后面会介绍)

功能：根据正则表达式扫描整个字符串，并返回所有能成功匹配的子串
    1）如果不能匹配成功，则返回一个空列表；
    2）如果能匹配成功，则返回包含所有匹配子串的列表
```

示例代码：

```
my_str = 'abc_123_DFG_456'

# 匹配字符串中的所有连续的3位数字
# raw string：原生字符串
res = re.findall(r'\d{3}', my_str)
print(type(res), res)
```

知识点8：re模块-split方法的使用【熟悉】

函数语法格式：

```
split函数：re.split(pattern, string, maxsplit=0, flags=0)
参数：
    pattern：匹配的正则表达式
    string：要进行分割的字符串
    maxsplit：最多分割次数，默认为0，表示分割所有
    flags：标志位(暂时忽略，后面会介绍)

功能：根据正则表达式匹配的子串对原字符串进行分割，返回分割后的列表
```

示例代码：

```
import re

my_str = '传智播客, Python, 数据分析'

# 需求：按照 `, ` 对上面的字符串进行分割
res = re.split(r',\s', my_str)
print(type(res), res)

res = re.split(r',\s', my_str, maxsplit=1)
print(type(res), res)

res = my_str.split(', ')
print(type(res), res)

my_str2 = '传智播客,Python;数据分析'

res = re.split(r'[,;]', my_str2)
print(type(res), res)
```

知识点9：re模块-正则分组操作【常握】

在使用正则表达式进行匹配操作时，可以将正则表达式进行分组，并在匹配之后获取相应分组的匹配数据。

正则匹配分组语法：

正则语法	描述
(正则表达式)	将括号中字符作为一个分组
(?P<name>正则表达式)	分组起别名

示例1：正则匹配分组操作

> 语法：(正则表达式)

```
import re

my_str = '13155667788'

# 需求：使用正则提取出手机号的前3位、中间4位以及后 4 位数据

res = re.match(r'(\d{3})(\d{4})(\d{4})', my_str)
print(type(res), res)

# 获取整个正则表达式匹配的内容
print(res.group())

# 获取正则表达式指定分组匹配到的内容
# Match对象.group(组号)
print(res.group(1)) # 131
print(res.group(2)) # 5566
print(res.group(3)) # 7788
```

示例2：给正则分组起别名

> 语法：(?P<分组别名>正则表达式)

```
my_str = '<div><a href="https://www.itcast.cn" target="_blank">传智播客</a>
<p>Python</p></div>'

# 需求：使用正则提取出 my_str 字符串中的 `传智播客` 文本

res = re.search(r'<a.*>(?P<text>.*)</a>', my_str)
print(type(res), res)

# 获取整个正则表达式匹配到的内容
print(res.group())

# 获取指定分组匹配到的内容
print(res.group(1)) # 传智播客

# 根据分组的名称，获取指定分组匹配到的内容
# Match对象.group(分组名称)
print(res.group('text')) # 传智播客
```

知识点10：re模块-引用正则分组【熟悉】

在正则表达式中，放在圆括号 `()` 中表示的是一个组，当使用 `()` 定义了一个正则表达式组后，正则引擎会将匹配到的组按照顺序编号，存入缓存，这样的话我们想在后面对已经匹配过的内容进行引用时，引用分组的方式如下：

正则语法	描述
`\num`	引用正则中第 num 个分组匹配到的字符串 例如： `\1` 表示第一个分组， `\2` 表示第二个分组.....
`(?P=name)`	引用正则中别名为 name 分组匹配到的字符串

分组引用示例代码：

```
# 需求：写一个正则表达式，匹配字符串形如：'xxx xxx xxx'
# 注意：xxx可以是任意多位的数字，但是这三个xxx必须一致，比如：'123 123 123', '6666 6666 6666'

import re

my_str = '123 123 123'

# 根据分组序号引用分组
res = re.match(r'(\d+)\s\1\s\1', my_str)
print(type(res), res)
# 获取整个正则表达式匹配到的内容
print(res.group())

print('=' * 20)

# 根据分组名称引用分组
res = re.match(r'(?P<num>\d+)\s(?P=num)\s(?P=num)', my_str)
print(type(res), res)
# 获取整个正则表达式匹配到的内容
print(res.group())
```

知识点11：re模块-sub方法使用【熟悉】

函数语法格式：

```
sub函数：re.sub(pattern, repl, string, count=0, flags=0)

参数：
    pattern：匹配的正则表达式
    repl：替换内容
    string：原字符串
    count：最多替换次数，默认为0，表示替换所有
    flags：标志位(暂时忽略，后面会介绍)

功能：根据正则表达式匹配字符串中的所有子串，然后使用指定内容进行替换
    1）函数返回的是替换后的新字符串
```

示例代码1：

```
import re
```

```
2
3   my_str = "传智播客-Python-666"
4
5   # 需求： 将字符串中的 - 替换成 _
6   new_str = re.sub(r'-', r'_', my_str)
7   print(type(new_str), new_str)
8
9   new_str = re.sub(r'-', r'_', my_str, count=1)
10  print(type(new_str), new_str)
11
12  new_str = my_str.replace('-', '_')
13  print(type(new_str), new_str)
14
15  my_str = "传智播客,Python;666"
16
17  # 需求： 将字符串中的 ,和; 替换成 _
18  new_str = re.sub(r'[,;]', r'_', my_str)
19  print(type(new_str), new_str)
```

示例代码2：

```
1   import re
2
3   # 需求：将字符串 `abc.123` 替换为 `123.abc`
4   my_str = 'abc.123'
5
6   new_str = re.sub(r'([a-z]{3})\.(\d{3})', r'\2.\1', my_str)
7   print(new_str)
```

知识点12：re模块-正则表示式修饰符【了解】

正则表达式可以包含一些可选标志修饰符来控制匹配的模式。

修饰符被指定为一个可选的标志。多个标志可以通过按位 OR(|) 它们来指定。如 re.I | re.M 被设置成 I 和 M 标志：

修饰符	描述
re.I	匹配时不区分大小写
re.M	多行匹配，影响 ^ 和 $
re.S	使 . 匹配包括换行在内的所有字符

示例代码：

```
1   import re
2
3   my_str = 'aB'
4
5   # re.I：匹配时不区分大小写
6   res = re.match(r'ab', my_str, flags=re.I)
7   print(bool(res)) # 非None就True，None就是False
8
9
10  print('=' * 20)
11  # 开启多行模式
12  # ^  可以匹配字符串开头（字符串的开始位置），也可以匹配行的开头（即换行符\n之后的位置）
```

```python
# $ 可以匹配字符串结尾（字符串的结束位置），也可以匹配行的结尾（即换行符\n之前的位置）

# 关闭多行模式
# ^ 只能匹配字符串开头
# $ 只能匹配字符串结尾
my_str = 'aabb\nbbcc'

res = re.findall(r'^[a-z]{4}$', my_str, flags=re.M)
print(res)
print(bool(res))
print('=' * 20)

my_str = '\nabc'
# re.S：影响 . 符号，设置之后，.符号就能匹配\n了
res = re.match(r'.', my_str, flags=re.S)
print(bool(res))
```

知识点13：贪婪模式和非贪婪模式【常握】

贪婪模式：在整个表达式匹配成功的前提下，尽可能多的匹配

非贪婪模式：在整个表达式匹配成功的前提下，尽可能少的匹配

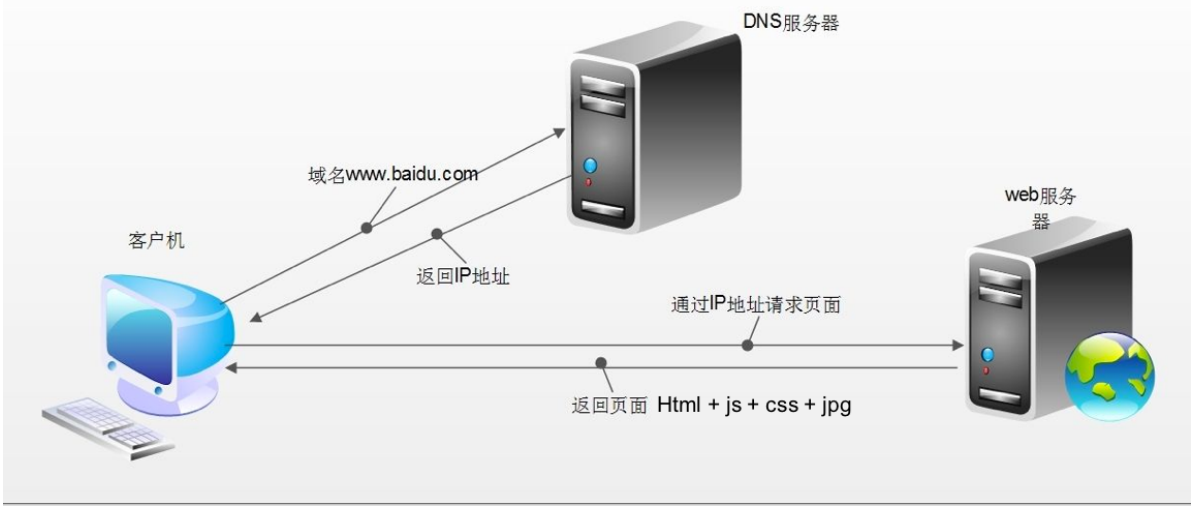
> 正则中的量词包括：{m,n}、?、*和+，这些量词默认都是贪婪模式的匹配，可以在这些量词后面加?将其变为非贪婪模式。
>
> 例如：\d{2,5}

示例代码：

```python
import re

my_str = '<div>test1</div><div>test2</div>'

# 贪婪模式：在整个表达式匹配成功的前提下，尽可能多的匹配
re_obj = re.match(r'<div>.*</div>', my_str)
print(re_obj)
print(re_obj.group()) # 获取整个正则表达式匹配的内容

# 非贪婪模式：在整个表达式匹配成功的前提下，尽可能少的匹配
re_obj = re.match(r'<div>.*?</div>', my_str)
print(re_obj)
print(re_obj.group()) # 获取整个正则表达式匹配的内容
```

知识点14：浏览器的请求过程-渲染【熟悉】

HTTP单次请求过程：



浏览器请求过程：

1. 浏览器通过域名解析服务器（DNS）获取IP地址
2. 浏览器先向IP发起请求，并获取响应
3. 在返回的响应内容（html）中，可能会带有css、js、图片等url地址，浏览器按照响应内容中的顺序依次发送其他的请求，并获取相应的响应
4. 浏览器每获取一个响应就对展示出的结果进行添加（加载），js、css等内容可能会修改页面的内容，js也可以重新发送请求，获取响应
5. 从获取第一个响应并在浏览器中展示，直到最终获取全部响应，并在展示的结果中添加内容或修改，**这个过程叫做浏览器的渲染**

知识点15：FastAPI-返回图片数据【掌握】

示例代码：

```python
# 导入 FastAPI 类
from fastapi import FastAPI
# 导入 uvicorn
import uvicorn
# 导入 Response 响应类
from fastapi import Response

# 创建 FastAPI 对象
app = FastAPI()


# 定义业务处理函数并设置对应的 URL 地址
# get：表示请求方式
# /index.html：表示请求的 URL 地址
@app.get('/index.html')
def index():
    with open('./sources/html/index.html', 'r', encoding='utf8') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='html')


@app.get('/gdp.html')
def gdp():
    with open('./sources/html/gdp.html', 'r', encoding='utf8') as f:
        content = f.read()

```

```python
    # 返回响应对象
    return Response(content, media_type='html')


@app.get('/images/0.jpg')
def get_images_0():
    with open('./sources/images/0.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


@app.get('/images/1.jpg')
def get_images_1():
    with open('./sources/images/1.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


@app.get('/images/2.jpg')
def get_images_2():
    with open('./sources/images/2.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


@app.get('/images/3.jpg')
def get_images_3():
    with open('./sources/images/3.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


@app.get('/images/4.jpg')
def get_images_4():
    with open('./sources/images/4.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


@app.get('/images/5.jpg')
def get_images_5():
    with open('./sources/images/5.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


```

```python
@app.get('/images/6.jpg')
def get_images_6():
    with open('./sources/images/6.jpg', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


if __name__ == '__main__':
    # 启动 Web 服务器
    uvicorn.run(app, host='127.0.0.1', port=8080)
```

知识点16：FastAPI-提取URL路径数据【掌握】

问题：

```
1. 对于上面的代码，假如我们有1000张图片要返回，该怎么操作？
```

通过观察，我们可以发现返回图片的代码都是相似的，只要能从 URL 路径中获得图片的名称，上面的返回图片处理函数就能融合成一个，在函数中只要根据图片名称读取对应的图片内容返回就可以了。

```
2. 如何从 URL 地址中提取数据？
```

FastAPI 可以从 URL 地址中提取数据，并将提取的数据传递给对应的处理函数，格式如下：

```python
from fastapi import FastAPI

app = FastAPI()


#@app.get('/.../{参数名}')
@app.get("/items/{item_id}")
def read_item(item_id): # FastAPI 提取了数据之后，会将提取的数据传递给下面处理函数的对应形参
    pass
```

动态返回图片数据示例代码：

```python
# 导入 FastAPI 类
from fastapi import FastAPI, Path
# 导入 uvicorn
import uvicorn
# 导入 Response 响应类
from fastapi import Response

# 创建 FastAPI 对象
app = FastAPI()


# 定义业务处理函数并设置对应的 URL 地址
# get：表示请求方式
# /index.html：表示请求的 URL 地址
@app.get('/index.html')
def index():
    with open('./sources/html/index.html', 'r', encoding='utf8') as f:
        content = f.read()

    # 返回响应对象
```

```
    return Response(content, media_type='html')


@app.get('/gdp.html')
def gdp():
    with open('./sources/html/gdp.html', 'r', encoding='utf8') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='html')


# 从 URL 地址中提取图片名称
@app.get('/images/{image_name}')
def get_images(image_name):
    print('image_name：', image_name)

    with open(f'./sources/images/{image_name}', 'rb') as f:
        content = f.read()

    # 返回响应对象
    return Response(content, media_type='jpg')


if __name__ == '__main__':
    # 启动 Web 服务器
    uvicorn.run(app, host='127.0.0.1', port=8080)
```

知识点17：爬虫的概念及工作过程【了解】

问题：

```
1. 什么是爬虫？ 爬虫有什么作用？
2. 爬虫工作过程是什么？
```

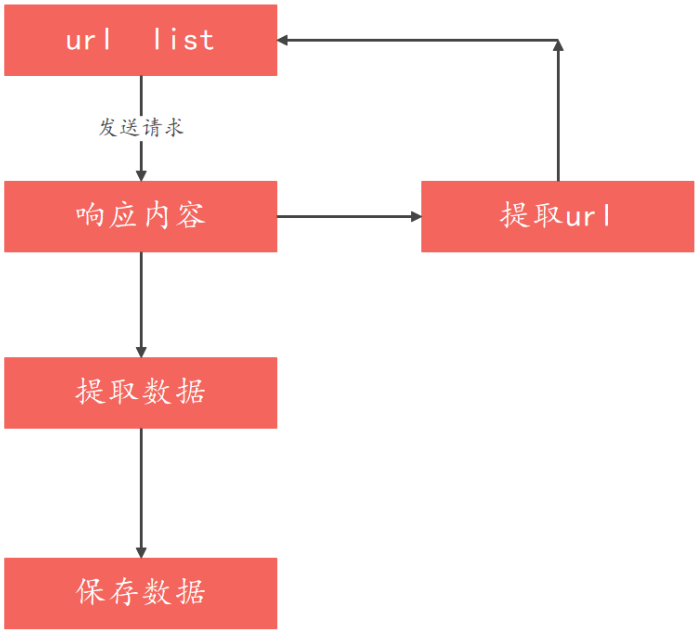
什么是爬虫？爬虫有什么作用？

网络爬虫（又被称为网页蜘蛛，网络机器人）就是模拟浏览器发送网络请求，接收请求响应，一种按照一定的规则，自动地抓取互联网信息的程序。

- 原则上，只要是浏览器(客户端)能做的事情，爬虫都能做。
- 爬虫只能获取到浏览器(客户端)所展示出来的数据。

爬虫的作用：数据分析中，进行数据采集的一种方式。

爬虫工作过程是什么？



1. 向起始的 url 地址发送请求，并获取响应数据
2. 对响应内容进行提取
3. 如果提取 url，则继续发送请求获取响应
4. 如果提取数据，将数据进行保存

知识点18：requests 模块的功能使用【掌握】

问题：

```
1. requests模块是什么？有什么作用？
2. 如果使用 requests 模块？
```

requests模块是什么？有什么作用？

requests 是用 python 语言编写的一个开源的HTTP库，可以通过 requests 库编写 python 代码发送网络请求，其简单易用，是编写爬虫程序时必知必会的一个模块。

中文文档： https://docs.python-requests.org/zh_CN/latest/index.html

如果使用 requests 模块？

1）安装

```
pip install requests
或者
pip install requests -i https://pypi.tuna.tsinghua.edu.cn/simple
```

2）示例：请求百度，并获取响应内容

```
"""
requests模块基本使用
学习目标：能够使用 requests 模块请求URL地址并获取响应内容
"""

# 导入 requests 包
import requests

# 准备 url 地址
url = 'https://www.baidu.com'
```

```python

# 使用 requests 发送 GET 请求
# 响应对象
response = requests.get(url)

# 获取响应的内容
# response.content：bytes，服务器返回的原始响应内容
# bytes -> str：bytes数据.decode('解码方式')
print(response.content.decode())
```