第4讲 内容解析与提取

慧科集团 李皓

主要内容

- 信息解析与提取的一般方法
- 不同类型信息的解析方法
  - 无结构
  - 半结构
  - 有结构
- 使用正则表达式解析文本
  - 语法
  - Python应用
- 使用XPATH解析XML类文本
  - Lxml
- python文本信息解析工具
  - BeautifulSoup

信息解析与提取的一般方法

1. 完整解析信息的标记形式，再提取关键信息；
   - 需要标记解析器；
   - 优点是解析准确；
   - 缺点是提取过程繁琐/速度慢。
2. 不解析全文，直接搜索信息；
   - 需要文本查找函数；
   - 优点是提取过程简洁，速度快；
   - 缺点是提取结果准确性和信息内容相关。
3. 适应性方法
   - 结合上述两种方法的方法。

2

不同类型信息的解析方法

页面内容分类

- 可分为三类：
  - 无结构的文本信息
    - 例如txt文本；
  - 半结构化的标记型文本信息，
    - 例如html网页、json数据、xml数据等；
  - 结构化的信息
    - 例如数据库文件、电子表格文件等；

信息的解析

- 针对无结构文本信息
  - 利用正则表达式进行模式匹配

待匹配的目标： abc123def456

正则表达式： [0-9]+ 或/d+

匹配结果： 123
456

文本信息的组织

- 流行的文本信息标记形式有三种：
  - XML
    - 可扩展标记语言；
    - 最早出现，扩展性好，但繁琐；
    - Internet信息交互和表达。
  - JSON
    - Javascript Object Notation
    - 使用有类型的键值对表达信息；
    - 适合于程序处理（js），较XML简洁；
    - 移动应用云端和节点的信息通信，无注释；
  - YAML
    - YAML Ain't Markup Language
    - 无类型的键值对表示信息；
    - 文本信息比例高，可读性好。
    - 系统配置信息

文本信息的组织

- XML实例

```
<person>
    <firstName>Tian</firstName>
    <lastName>Song</lastName>
    <address>
        <streetAddr>中关村南大街5号</streetAddr>
        <city>北京市</city>
        <zipcode>100081</zipcode>
    </address>
    <prof>Computer System</prof><prof>Security</prof>
</person>
```

文本信息的组织

- JSON 实例

```
{
    “firstName” : “Tian” ,
    “lastName”  : “Song” ,
    “address”   : {
                    “streetAddr” : “中关村南大街5号” ,
                    “city”       : “北京市” ,
                    “zipcode”    : “100081”
                  } ,
    “prof”      : [ “Computer System” , “Security” ]
}
```

文本信息的组织

- YAML实例

```
firstName : Tian
lastName  : Song
address   :
    streetAddr : 中关村南大街5号
    city       : 北京市
    zipcode    : 100081
prof          :
-Computer System
-Security
```

信息的解析

- 针对半结构化文本信息
  - 针对HMTL文档，有下列方法
    - 利用正则表达式进行模式匹配
    - 利用xpath进行HTML标签检索
    - 利用CSS选择器进行HTML标签检索
  - 针对JSON数据
    - 利用JSON Path进行检索
    - 利用Python类型转换为json类
  - 针对XML数据
    - 转化成Python类型（xmltodict）
    - XPath
    - CSS选择器
    - 正则表达式

信息的解析

- 结构化数据的解析
  - 针对各类数据库文件，可以借助相关的python库
    - 对于Mysql数据库文件，可以使用pymysql库；
    - 对于Sqllite数据库文件，可以使用Sqlite3库；
    - 对于MS SqlServer数据库文件，可以使用pyodbc+pymssql库；
    - 对于Orange数据库文件，可以使用cx_oracle库；

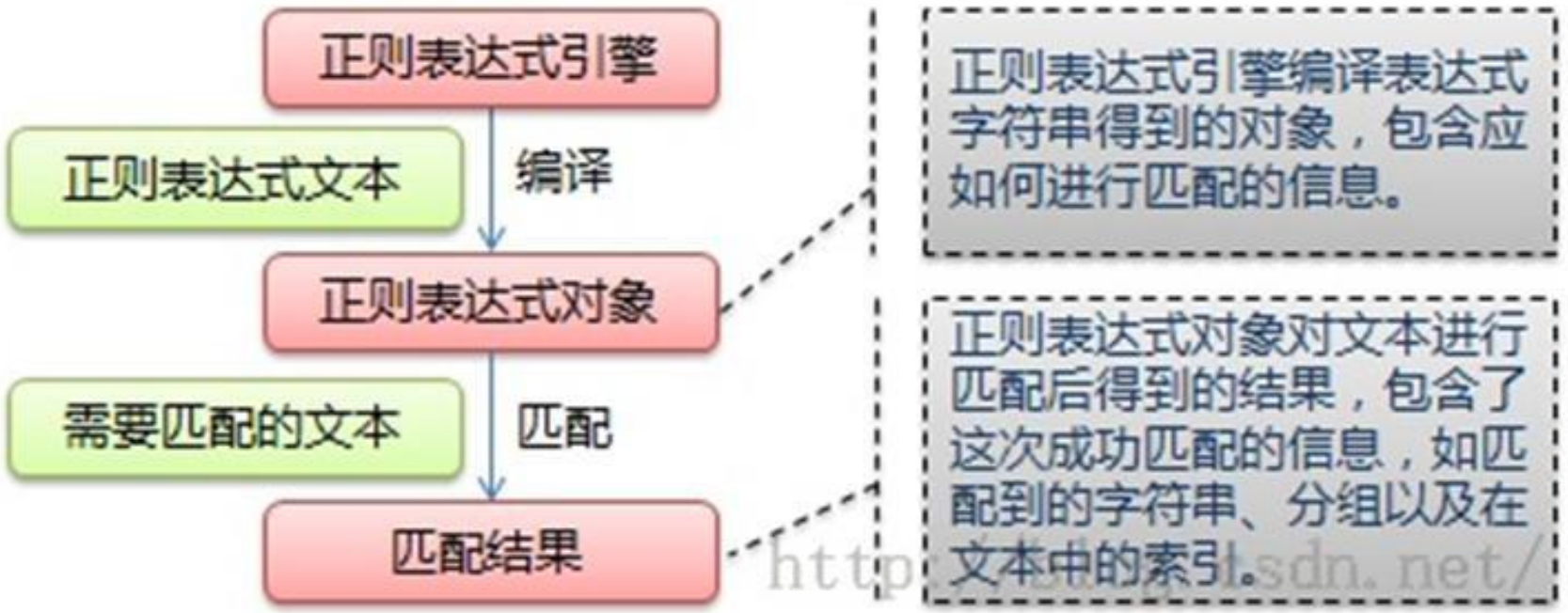
信息的解析

- 结构化数据的解析
  - 针对excel文件
    - xlwings：简单强大，可替代VBA；
    - openpyxl：简单易用，功能广泛；
    - pandas：数据处理功能强大；
    - win32com：还可以处理office其他类型文件;
    - Xlsxwriter：易于生成Excel文档；
    - DataNitro：内嵌于excel中，可替代VBA；
    - xlutils：结合xlrd/xlwt使用。

3

正则表达式

正则表达式Regular Expression



正则表达式匹配规则

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符"\n"外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符*需要匹配，可以使用*或者字符集[*]。	a\.c a\\c	a.c a\c
[...]	字符集（字符类）。对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用]、-或^，可以在前面加上反斜杠，或把]、-放在第一个字符，把^放在非第一个字符。	a[bcd]e	abe ace ade
预定义字符集（可以写在字符集[...]中）			
\d	数字：[0-9]	a\dc	a1c
\D	非数字：[^\d]	a\Dc	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\sc	a c
\S	非空白字符：[^\s]	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（用在字符或(...)之后）			
*	匹配前一个字符0或无限次。	abc*	ab abccc

正则表达式匹配规则

数量词（用在字符或(...)之后）			
*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	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{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n}变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配\w和\W之间。	a\b!bc	a!bc
\B	[^\b]	a\Bbc	abc

正则表达式匹配规则

逻辑、分组			
|	|代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果|没有被包括在()中，则它的范围是整个正则表达式。	abc|def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号'('，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的|仅在该组中有效。	(abc){2} a(123|456)c	abcabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?:...)	(...)的不分组版本，用于使用'|'或后接数量词。	(?:abc){2}	abcabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern |no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 |no-patern可以省略。	(\d)abc(?(1)\d|abc)	1abc2 abcabc

正则表达式-实例

- 匹配中文
  - [\u4e00-\u9fa5]
- 匹配双字节字符（含汉字）
  - [^\x00-\xff]
- 匹配空白行
  - \n\s*\r
- 匹配Email
  - [\w!#$%&'*+/=?^_`{|}~-]+(?:\.[\w!#$%&'*+/=?^_`{|}~-]+)*@(?:[\w](?:[\w-]*[\w])?\.)+[\w](?:[\w-]*[\w])?

正则表达式-实例

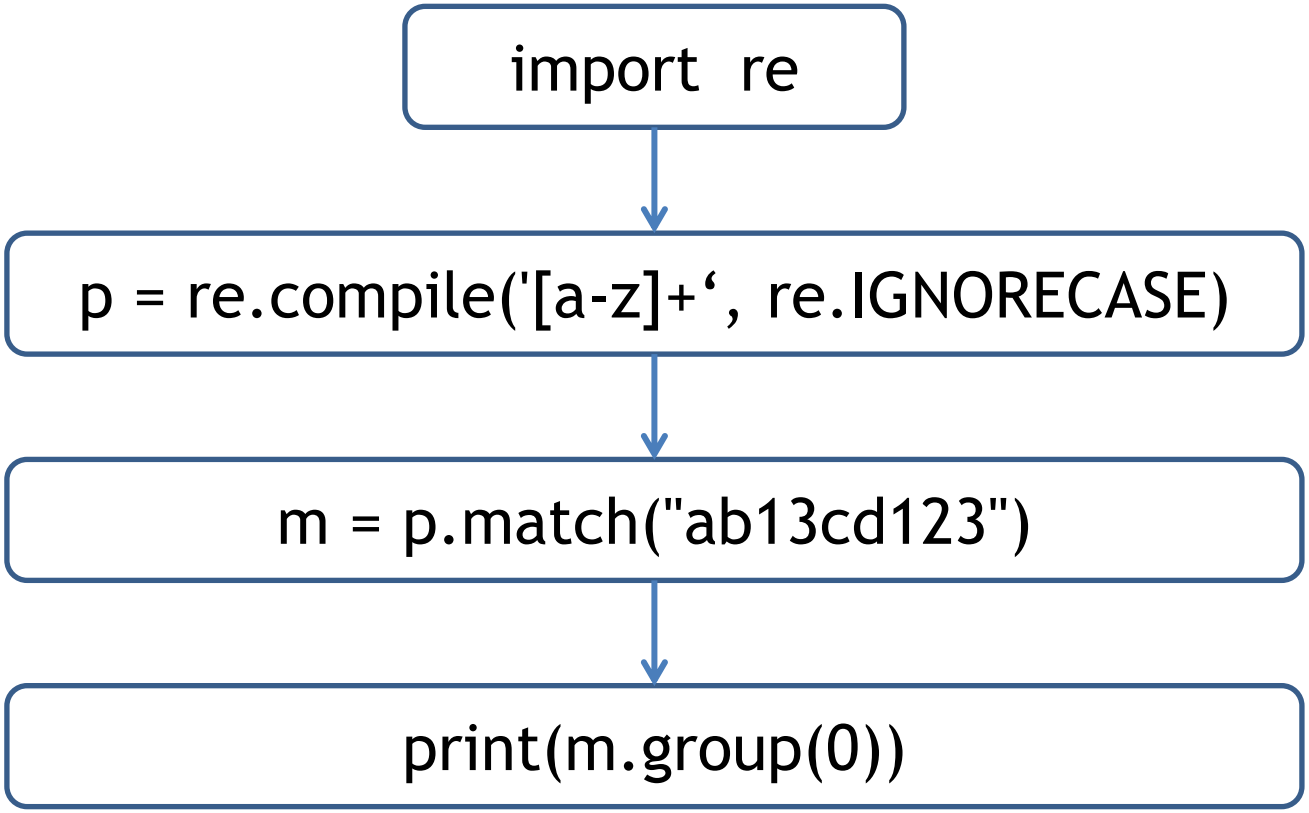
- 匹配URL
  - [a-zA-z]+://[^\s]*
- 匹配国内电话号码
  - \d{3}-\d{8}|\d{4}-\{7,8}
- 匹配腾讯QQ号
  - [1-9][0-9]{4,}
- 匹配中国邮政编码
  - [1-9]\d{5}(?!\d)
- 匹配18位身份证号
  - ^(\d{6})(\d{4})(\d{2})(\d{2})(\d{3})([0-9]|X)$

正则表达式-实例

- 匹配年-月-日格式日期
  - ([0-9]{3}[1-9]|[0-9]{2}[1-9][0-9]{1}|[0-9]{1}[1-9][0-9]{2}|[1-9][0-9]{3})-(((0[13578]|1[02])-(0[1-9]|[12][0-9]|3[01]))|((0[469]|11)-(0[1-9]|[12][0-9]|30))|(02-(0[1-9]|[1][0-9]|2[0-8])))
- 匹配负整数
  - ^-[1-9]\d*$
- 匹配整数
  - ^-?[1-9]\d*$
- 匹配正浮点数
  - ^[1-9]\d*\.\d*|0\.\d*[1-9]\d*$
- 匹配负浮点数
  - ^-[1-9]\d*\.\d*|-0\.\d*[1-9]\d*$

正则表达式Regular Expression

- python



4

使用XPATH解析XML类文本

XML

- 节点关系
  - 父（Parent)
  - 子（Children)
  - 同胞（Sibling)
  - 先辈（Ancestor)
  - 后代（Descendant

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

    <book>
      <title lang="eng">Harry Potter</title>
      <price>29.99</price>
    </book>

    <book>
      <title lang="eng">Learning XML</title>
      <price>39.95</price>
    </book>

</bookstore>
```

XPATH

- XPath (XML Path Language) 是一门在 XML 文档中查找信息的语言，可用来在 XML 文档中对元素和属性进行遍历。