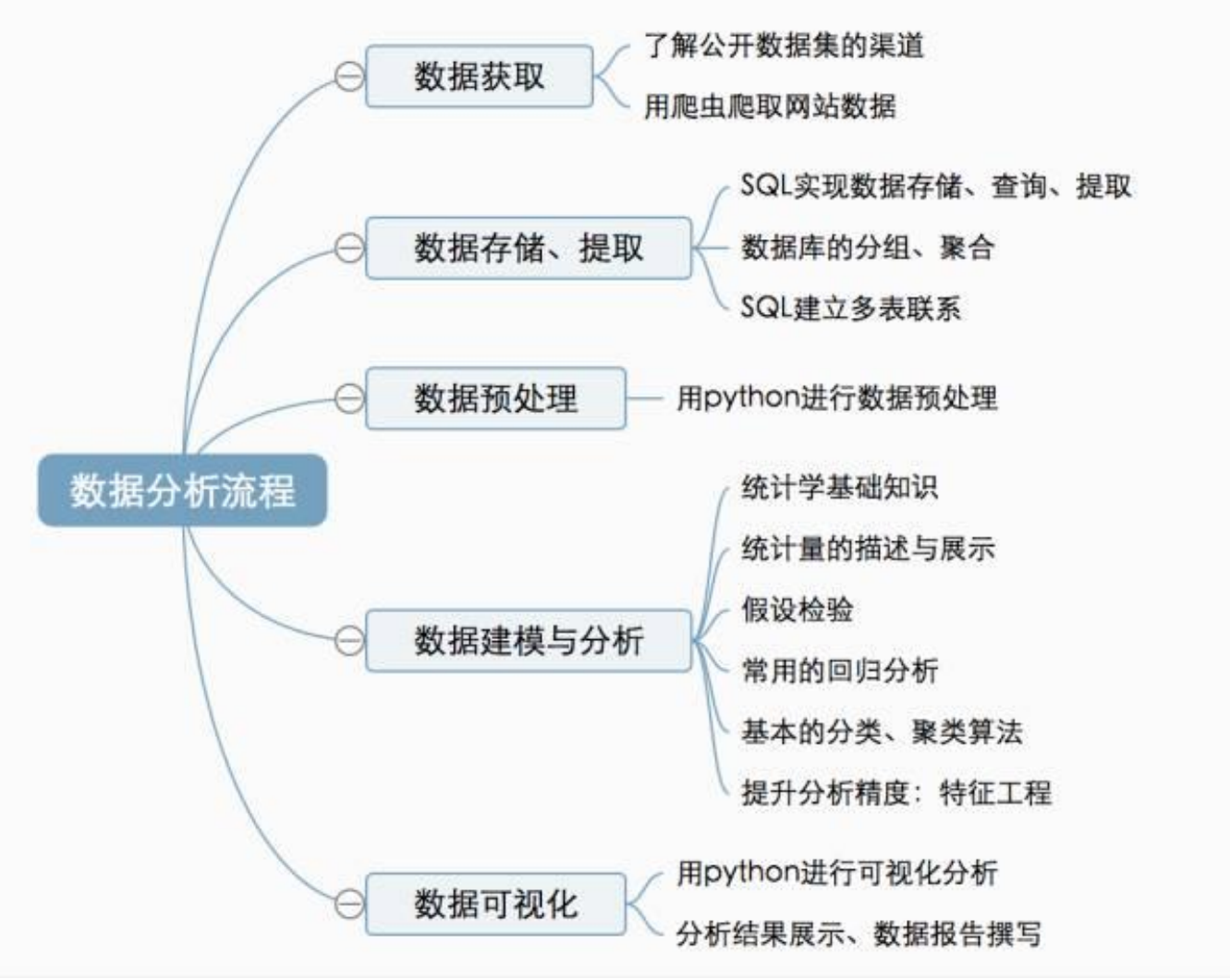
课程介绍

课程介绍


数据分析流程
数据获取
了解公开数据集的渠道
用爬虫爬取网站数据
数据存储、提取
SQL实现数据存储、查询、提取
数据库的分组、聚合
SQL建立多表联系
数据预处理
用python进行数据预处理
数据建模与分析
统计学基础知识
统计量的描述与展示
假设检验
常用的回归分析
基本的分类、聚类算法
提升分析精度：特征工程
数据可视化
用python进行可视化分析
分析结果展示、数据报告撰写

数据采集情景

- 日志采集
  - 日志信息抓取
  - 日志信息归档
  - 日志信息结构化存储
- 非结构化数据采集
  - 非结构化数据文件到结构化数据库
  - 非结构化数据文件到NOSQL
  - 结构化数据到非结构化数据文件
  - 非结构化数据变化采集(pyinotify)

数据采集情景

- 网络采集
  - RPC 数据采集
  - 远程socket采集
  - 数据库采集
- 工业及物联网数据采集
  - 协议部分
  - 架构部分
  - 展示部分
  - 存储部分
- 爬虫采集
  - 网页数据采集
  - XML/JSON 数据采集
  - RESULTFUL 数据采集

工具介绍

开发工具

◆编辑器：

notepad++

Sublime

Atom

Pycharm

◆上传

Winscp

Filezilla

◆运行

python

Ipython

Jupyter

pycharm

jupyter

```
yum -y install python-devel
pip install jupyter
```

前台运行

```
jupyter notebook --allow-root --port 6001 --ip=10.110.13.186
```

后台运行

```
nohup jupyter notebook --allow-root --port 6001 --ip=10.110.13.186 1>/dev/null 2>&1 &
```

浏览器访问

http://10.110.13.186:6001/tree?

jupyter

jupyter Untitled1 Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [2]:
```
import tushare as ts
e = ts.get_hist_data('601318',start='2017-06-23',end='2017-06-26')
print e
```

```
             open   high  close    low     volume  price_change  p_change  \
date
2017-06-26  49.99  50.77  49.72  49.35  953192.56          0.00      0.00
2017-06-23  49.24  49.90  49.72  48.79  873719.62          0.58      1.18

                ma5    ma10    ma20      v_ma5     v_ma10     v_ma20
date
2017-06-26  49.222  48.320  47.288  887897.74  860166.06  857338.21
2017-06-23  48.990  48.241  47.076  858955.61  850806.49  879173.58
```

In [3]:
```
import tushare as ts
df = ts.get_index()
print(df)
```

```
      code    name  change        open    preclose       close        high  \
0   000001    上证指数   -0.31   2879.6901   2882.2254   2873.3557   2882.1249
1   000002    A股指数   -0.31   3015.9356   3018.6076   3009.2726   3018.4836
2   000003    B股指数    0.22    293.4023    293.0929    293.7331    294.0626
3   000008    综合指数   -0.35   2615.0480   2618.7486   2609.5811   2616.8268
4   000009   上证380   -0.28   4830.1413   4827.7587   4814.2100   4833.5879
5   000010   上证180   -0.29   7688.0953   7695.2814   7672.6756   7699.1513
6   000011    基金指数   -0.09   5880.0829   5881.1888   5876.0431   5883.6086
7   000012    国债指数    0.00    166.2860    166.2700    166.2727    166.2871
```

组件库安装

◆在线安装

```
pip install requests
```

◆在线whl安装

```
https://pypi.org/project/requests/#files
pip install requests-2.19.1-py2.py3-none-any.whl
```

◆源码安装

```
tar zxvf requests-2.19.1.tar.gz
```

或者

```
git clone https://github.com/requests/requests.git
python setup.py build
python setup.py install
```

XML介绍

XML介绍

- **XML 指可扩展标记语言**
- **XML 被设计用来传输和存储数据。**

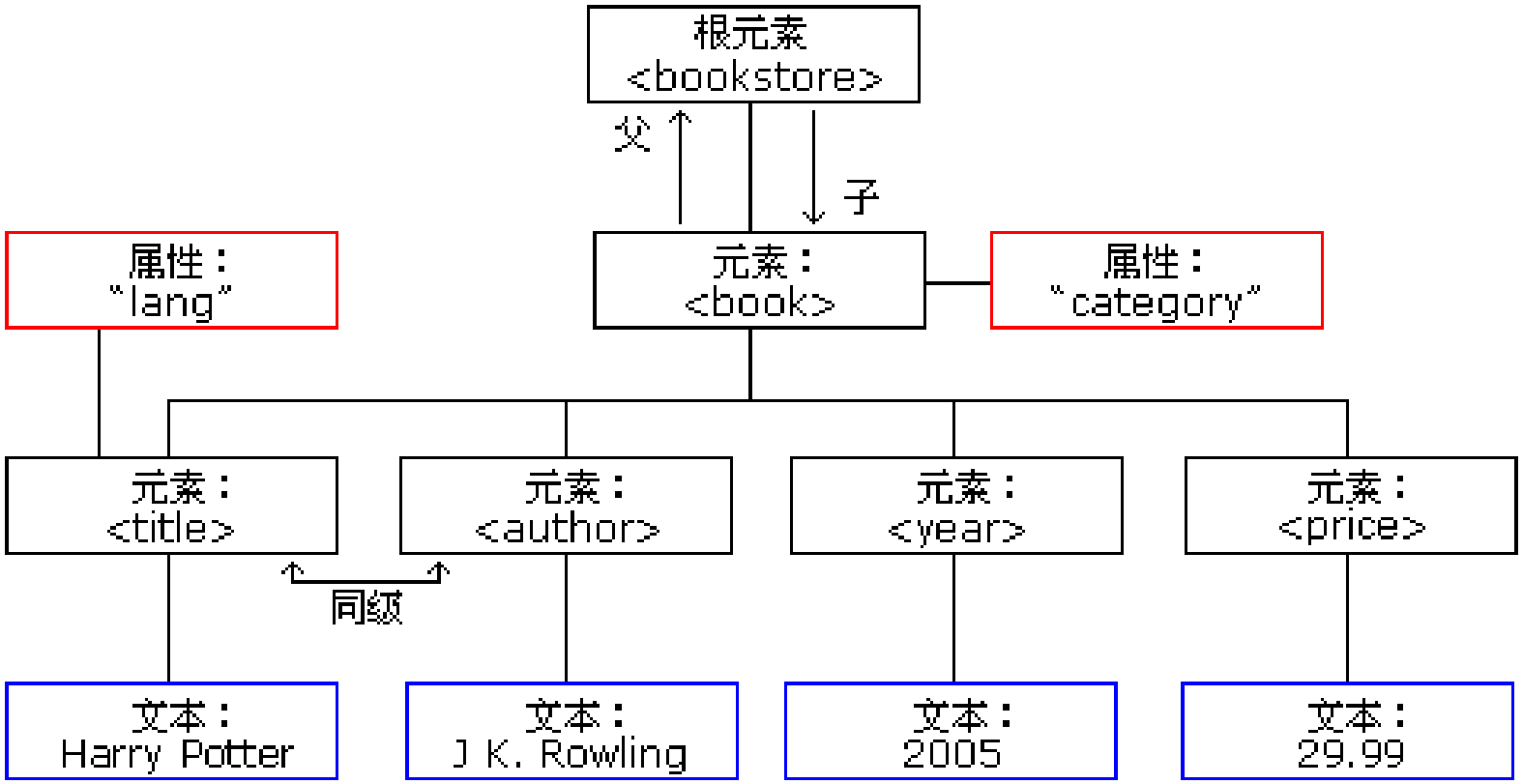
用途：

- 用 XML Schema 定义 XML 的结构和数据类型
- 用 XSLT 来转换 XML 数据
- 用 SOAP 来交换应用程序之间的 XML 数据
- 用 WSDL 来描述网络服务
- 用 RDF 来描述网络资源
- 用 XPath 和 XQuery 来访问 XML 数据
- 用 SMIL 来定义图形

示例

```
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

树结构


根元素
<bookstore>
父
子
属性：
"lang"
元素：
<book>
属性：
"category"
元素：
<title>
元素：
<author>
元素：
<year>
元素：
<price>
同级
文本：
Harry Potter
文本：
J K. Rowling
文本：
2005
文本：
29.99

XML的节点关系

- 根节点（root）
- 父(Parent)
- 子(Children)
- 同胞(Sibling)
- 先辈(Ancestor)
- 后代
- 属性

XML语法

- 所有 XML 元素都须有关闭标签
- XML 标签对大小写敏感
- XML 必须正确地嵌套
- XML 文档必须有根元素
- XML 的属性值须加引号
- 实体引用（< > & ’ ″ ）
- XML 中的注释
- 在 XML 中，空格会被保留
- XML 以 LF 存储换行

XML 验证 - DTD

◆DTD 的作用是定义 XML 文档的结构。它使用一系列合法的元素来定义文档结构：

```
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
```

XML 验证 - XML SCHEMA

◆W3C 支持一种基于 XML 的 DTD 代替者，它名为 XML Schema：

```
<xs:element name="note">
<xs:complexType>
  <xs:sequence>
    <xs:element name="to"      type="xs:string"/>
    <xs:element name="from"    type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body"    type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

XML 验证

- 用浏览器打开
- 用xmlspy打开
- 在线验证
- 根据 DTD 来验证 XML
- 根据xml schema验证

XML-解析-DOM

1)DOM解析

DOM是html和xml的应用程序接口(API)，以层次结构（类似于树型）来组织节点和信息片段，映射XML文档的结构，允许获取

和操作文档的任意部分，是W3C的官方标准

【优点】

①允许应用程序对数据和结构做出更改。

②访问是双向的，可以在任何时候在树中上下导航，获取和操作任意部分的数据。

【缺点】

①通常需要加载整个XML文档来构造层次结构，消耗资源大。

XML-解析-SAX

SAX(Simple API for XML)解析

流模型中的"推"模型分析方式。通过事件驱动，每发现一个节点就引发一个事件，事件推给事件处理器，通过回调方法

完成解析工作，解析XML文档的逻辑需要应用程序完成

【优势】

①不需要等待所有数据都被处理，分析就能立即开始。

②只在读取数据时检查数据，不需要保存在内存中。

③可以在某个条件得到满足时停止解析，不必解析整个文档。

④效率和性能较高，能解析大于系统内存的文档。

【缺点】

①需要应用程序自己负责TAG的处理逻辑（例如维护父/子关系等），文档越复杂程序就越复杂。

②单向导航，无法定位文档层次，很难同时访问同一文档的不同部分数据，不支持XPath。

XML-解析-STAX

StAX(Streaming API for XML)

流模型中的拉模型分析方式。提供基于指针和基于迭代器两种方式的支持。

【和推式解析相比的优点】

①在拉式解析中，事件是由解析应用产生的，因此拉式解析中向客户端提供的是解析规则，而不是解析器。

②同推式解析相比，拉式解析的代码更简单，而且不用那么多库。

③拉式解析客户端能够一次读取多个XML文件。

④拉式解析允许你过滤XML文件和跳过解析事件。

XML-解析-Stream

◆和stax类似，不同语言称呼不同

QT:

QXMLStreamReader,读XML文件实例

QXMLStreamWriter,写XML文件实例

Android:

pull解析，也是官方解析布局文件所使用的方式。Pull与SAX有点类似，都提供了类似的事件，如开始元素和结束元素。不同的是，SAX的事件驱动是回调相应方法，需要提供回调的方法，而后在SAX内部自动调用相应的方法。而Pull解析器并没有强制要求提供触发的方法。因为他触发的事件不是一个方法，而是一个数字。它使用方便，效率高。

XML-XSLT

◆通过使用 XSLT，您可以向 XML 文档添加显示信息。

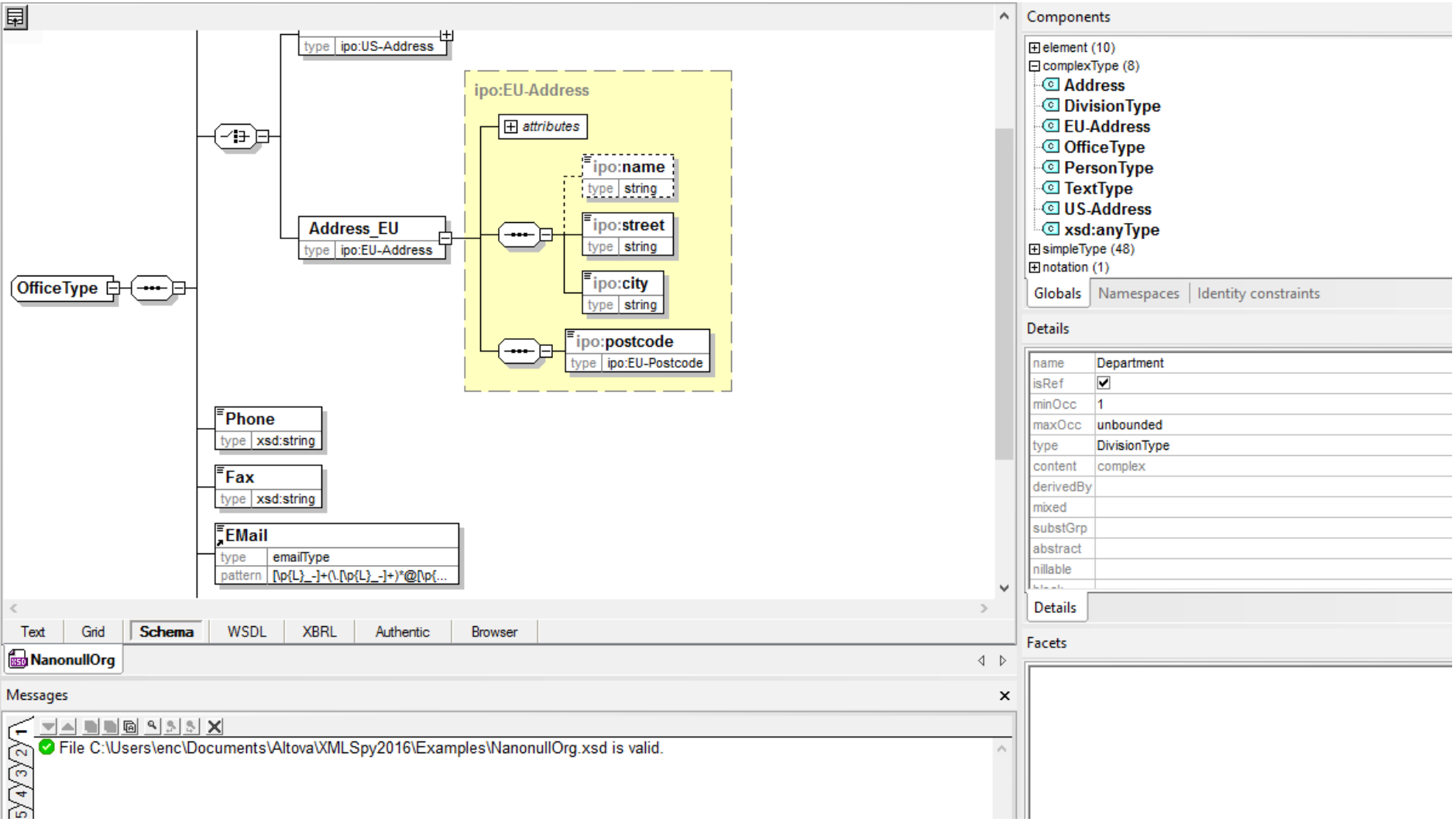
XSL = XML 样式表

XML 不使用预先定义的标签（我们可以使用任何喜欢的标签名），并且这些标签的意义并不都那么容易被理解。

<table> 元素意味着一个 HTML 表格，一件家具，或是别的什么东西 - 浏览器不清楚如何显示它。

XSL 可描述如何来显示 XML 文档！

XML 编辑器-xmlspy



XMLRPC

```
#helloserver.py

from SimpleXMLRPCServer import SimpleXMLRPCServer

def hello():
    print "hello,world!"
    return "hello client from server!"

svr=SimpleXMLRPCServer(("", 8081), allow_none=True)
svr.register_function(hello)
svr.serve_forever()
```

XML文件解析

- 转化为Python类型（xmltodict）
- XPath
- CSS选择器
- 正则表达式

XMLRPC

```
from xmlrpclib import ServerProxy
svr=ServerProxy("http://10.110.13.186:8081")
print svr.hello()
```

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中对元素和属性进行遍历。
- **XPath 开发工具**

开源的XPath表达式编辑工具：XML Quire(XML格式文件可用)

Chrome插件Xpath Helper

Firefox插件Xpath Checker

选取节点

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。 注释：假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

Lxml-xpath

```
from lxml import etree
root = etree.parse("bookstore.xml")
find = etree.XPath('//book')
allbook=find(root)
print(dir(allbook[0]))
for book in allbook:
    print(book.find('title').text)
for book in allbook:
    print(book.xpath('title/@lang'))
find = etree.XPath('/bookstore/book')
allbook=find(root)
for book in allbook:
    print(book.find('title').text)
find = etree.XPath('//@lang')
alllang=find(root)
for lang in alllang:
    print(lang)
```

lxml库

- lxml是一个HTML/XML的解析器，主要的功能是如何提取和解析HTML/XML数据。
- lxml和正则一样，也是用C实现，是一款高性能的 Python HTML/XML解析器，我们可以利用之前学习的XPath语法，来快速的定位特定元素以及节点信息。
- lxml python官方文档：http://lxml.de/index.html
- 需要安装C语言库，可使用pip安装：pip install lxml（或通过wheel方式安装）

Lxml xml加载

- 从字符串构建

```
xml_data = '<html>html.text<body>wo ai ni<child>child.text</child>tails</body></html>'
root1=etree.fromstring(xml_data)
```

- 从字符串构建2

```
root1 = etree.XML("<root><a>TEXT</a></root>")
```

- 字符串构建3

```
>>> doc = etree.fromstring(xmldata)
```

- 从文件构建

```
Doc=etree.parse( 'filename' )
```

- 设置解析器

```
parser= etree.XMLParser(remove_blank_text=True) #去除xml文件里的空行
root = etree.XML("<root>  <a/>   <b>  </b>     </root>",parser)
print etree.tostring(root)
```

Lxml xpath 例子

```
In [49]: root = etree.XML("<root><a><b/></a><b>bbb</b></root>")

In [50]:

In [50]: find = etree.XPath("//b")

In [51]:

In [51]: print(find(root)[0].tag)
b

In [52]: print(find(root)[1].text)
bbb
```

Lxml xpath 例子

```
#xpath_li.py

from lxml import etree

html = etree.parse('hello.html')
print type(html) #显示etree.parse()返回类型

result = html.xpath('//li')

print result  #打印<li>标签的的元素集合
print len(result)
print type(result)
print type(result[0])
```

Lxml xpath 函数使用

```
In [54]: count_elements = etree.XPath("count(//*[local-name() = $name])")

In [55]: print(count_elements(root, name = "a"))
1.0

In [56]: print(count_elements(root, name = "b"))
2.0

authors = etree.XPath( “string-join(//text() “,’ ,’ )
```

BeautifulSoup4

和lxml一样，Beautiful Soup也是一个HTML/XML的解析器，主要的功能也是如何解析和提取HTML/XML数据。

lxml只会局部遍历，而Beautiful Soup是基于HTML DOM的，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多，所以性能要低于lxml。

BeautifulSoup用来解析HTML比较简单，API非常人性化，支持CSS选择器、Python标准库中的HTML解析器，也支持lxml的XML解析器。

Beautiful Soup3目前已经停止开发，推荐现在的项目使用Beautiful Soup。

使用pip安装即可：pip install beautifulsoup4

搜索文档树

1. find_all(name, attrs, recursive, text, **kwargs)

1) name参数

name参数可以查找所有民资为name的tag，字符串对象会自动忽略掉。

A.传字符串

最简单的过滤器是字符串，在搜索方法中传入一个字符串参数，eautiful Soup会自动查找与字符串完整匹配的内容，下面的例子用于查找文档中所有的标签：

```
soup.find_all('b')
#[<b>The Dormouse's story</b>]

print soup.find_all('a')
#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>, <a
class="sister" href="http://example.com/lacie" id="link2">Lacie</a>, <a class="sister"
href="http://example.com/tillie" id="link3">Tillie</a>]
```

搜索文档树

B.传正则表达式

如果传入正则表达式作为参数，Beautiful Soup会通过正则表达式的match()来匹配内容。下面例子中找出所有以b开头的标签，这表示 `<body>` 和 `<b>` 标签都应该被找到。

```
import re
for tag in soup.find_all(re.compile('^b')):
  print(tag.name)

#body
#b
```

搜索文档树

C.传列表

如果传入列表参数，Beautiful Soup会将与列表中任一元素匹配的内容返回 下面代码找到文档中所有 `<a>` 标签和 `<b>` 标签：

```
soup.find_all(['a', 'b'])

# [<b>The Dormouse's story</b>,
#  <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

搜索文档树

2) keyword参数

```
soup.find_all(id='link2')
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

3) text参数

通过text参数可以搜索文档中的字符串内容，与name参数的可选值一样，text参数接收参数值，正则表达式，列表

```
soup.find_all(text='Elsie')
#[u'Elsie']

soup.find_all(text=['Tillie', 'Elsie', 'Lacie'])
# [u'Elsie', u'Lacie', u'Tillie']

soup.find_all(text=re.compile("Dormouse"))
[u"The Dormouse's story", u"The Dormouse's story"]
```

JSON

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式，它是的人们很容易的进行阅读和编写。同时也方便了机器进行解析和生成。适用于进行数据交互的场景，比如网站前台与后台之间的数据交互。

JSON和XML的比较可谓不相上下。

Python中自带了JSON模块，直接import json就可以使用了。

Json结构

JSON

json简单说就是javascript中的对象和数组，所以这两种结构就是对象和数据两种结构，通过这两种结构可以表示各种复杂的结构。

1. 对象：对象在js中表示为{}括起来的内容，数据结构为{key:value,key:value,...}的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为对象.key获取属性值，这个属性值的类型可以是数字、字符串、数组、对象这几种。
2. 数组：数组在js中是中括号[]括起来的内容，数据结构为["Python", "javascript", "C++",..]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是数字、字符串、数组、对象几种。

Json与python类型对应

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
true	True
false	False
null	None

Xpath与jsonpath

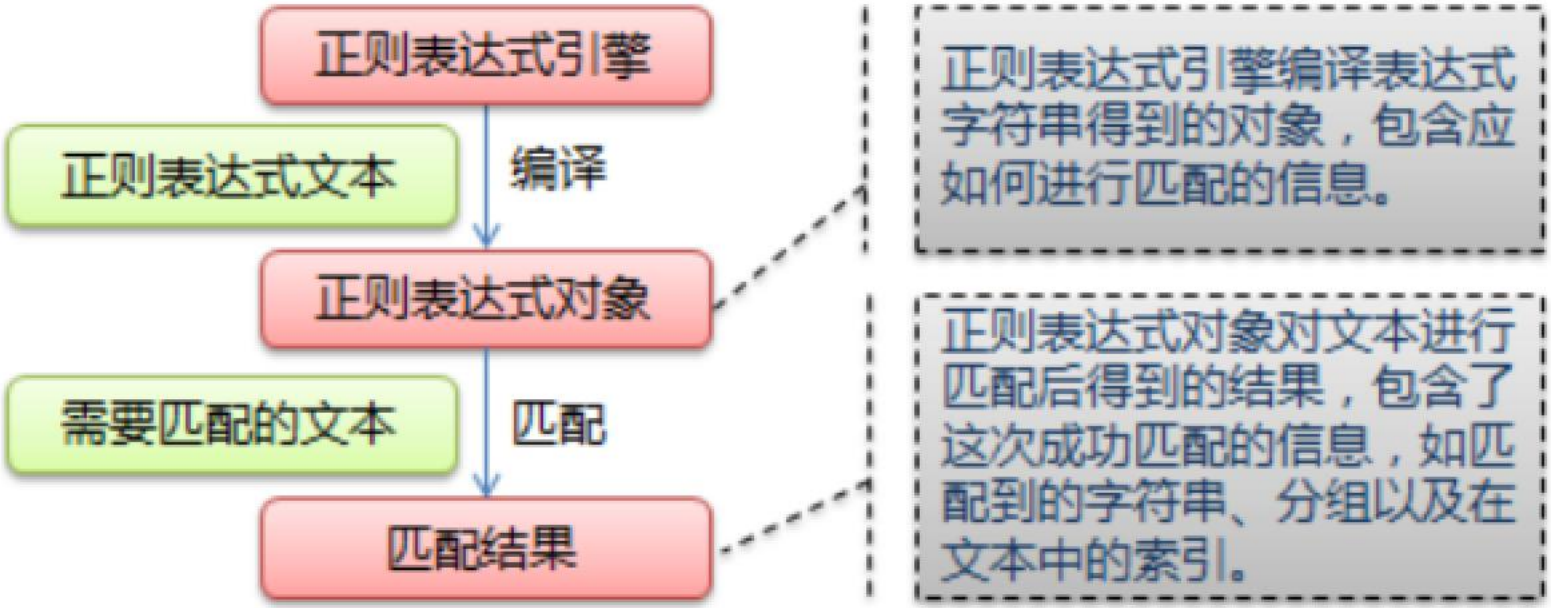
XPath	JSONPath	Description
/	$	the root object/element
.	@	the current object/element
/	. or []	child operator
..	n/a	parent operator
//	..	recursive descent. JSONPath borrows this syntax from E4X.
*	*	wildcard. All objects/elements regardless their names.
@	n/a	attribute access. JSON structures don't have attributes.
[]	[]	subscript operator. XPath uses it to iterate over element collections and for predicates. In Javascript and JSON it is the native array operator.
|	[,]	Union operator in XPath results in a combination of node sets. JSONPath allows alternate names or array indices as a set.
n/a	[start:end:step]	array slice operator borrowed from ES4.
[]	?()	applies a filter (script) expression.
n/a	()	script expression, using the underlying script engine.
()	n/a	grouping in Xpath

Xpath与jsonpath

XPath	JSONPath	Result
/store/book/author	$.store.book[*].author	the authors of all books in the store
//author	$..author	all authors
/store/*	$.store.*	all things in store, which are some books and a red bicycle.
/store//price	$.store..price	the price of everything in the store.
//book[3]	$..book[2]	the third book
//book[last()]	$..book[(@.length-1)] $..book[-1:]	the last book in order.
//book[position()<3]	$..book[0,1] $..book[:2]	the first two books
//book[isbn]	$..book[?(@.isbn)]	filter all books with isbn number
//book[price<10]	$..book[?(@.price<10)]	filter all books cheapier than 10
//*	$..*	all Elements in XML document. All members of JSON structure.

正则表达式

Python 正则表达式



Python 正则表达式规则

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
+	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
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	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。	(\d)abc(?(1)\d|abc)	1abc2 abcabc

正则表达式举例

```
>>>import re
>>>pattern = re.compile('\d+')
>>>m = pattern.search('one12twothree34four')  #这里如果使用match方法则不匹配
>>>m
<_sre.SRE_Match object at 0x10cc03ac0>
>>>m.group()
'12'
>>>m = pattern.search('one12twothree34four', 10, 30)   #指定字符串区间
>>>m
<_sre.SRE_Match object at 0x10cc03b28>
>>>m.group()
'34'
>>>m.span()
(13, 15)
```

开发库介绍

开发库介绍

Urllib

Httplib

Requests

lxml

Beautifulsoup4

Pycurl

selenium

jinja2

Flask

Ftplib

Telnetlib

paramiko

urllib

◆简单的抓取网页

```
import urllib
import urllib2
url="http://www.inspur.com"
request = urllib2.Request(url)
response = urllib2.urlopen(request)
print(response.read())
```

urllib

Jupyter

In [11]:
```
import urllib
import urllib2
from bs4 import BeautifulSoup

url="http://www.inspur.com"
request = urllib2.Request(url)
response = urllib2.urlopen(request)

body=response.read()
soup = BeautifulSoup(body,"html.parser")
imgs= soup.find_all('img', attrs={"border": "0"})
for img in imgs:
    print img['src']
```

```
/lcjtww/resource/cms/2018/03/img_pc_site/20180314182555259O5.png
/lcjtww/resource/cms/2018/03/img_pc_site/20180314182347599T3.png
/lcjtww/resource/cms/2018/03/img_pc_site/20180314182322266Q5.png
/lcjtww/resource/cms/2018/06/img_pc_site/20180616182452552.jpg
/lcjtww/resource/cms/2018/06/img_pc_site/20180604134755637Q1.jpg
/lcjtww/resource/cms/2017/12/img_pc_site/20171207113909108B9.jpg
```

urllib

◆直接将URL保存为本地文件

```
import urllib.request
url="http://www.xxxx.com/1.jpg"
urllib.request.urlretrieve(url,r"d:\temp\1.jpg")
```

urllib

◆带参数

```
import urllib.parse
import urllib.request
url="http://www.google.cn/webhp"
values={"rls":"ig"}
data=urllib.parse.urlencode(values)
theurl=url+"?"+data
#创建请求对象
req=urllib.request.Request(theurl)
#获得服务器返回的数据
response=urllib.request.urlopen(req)
#处理数据
page=response.read()
```

Urllib-post

◆带内容

```
import urllib.parse
import urllib.request
url="http://liuxin-blog.appspot.com/messageboard/add"
values={"content":"命令行发出网页请求测试"}
data=urllib.parse.urlencode(values)

#创建请求对象
req=urllib.request.Request(url,data)
#获得服务器返回的数据
response=urllib.request.urlopen(req)
#处理数据
page=response.read()
```

Httplib-get

```
import httplib
conn = httplib.HTTPConnection("www.python.org")
conn.request("GET", "/index.html")
r1 = conn.getresponse()
print r1.status, r1.reason
200 OK
data1 = r1.read()
conn.request("GET", "/parrot.spam")
r2 = conn.getresponse()
print r2.status, r2.reason
404 Not Found
data2 = r2.read()
conn.close()
```

Httplib-post

```
import httplib, urllib
params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
headers = {"Content-type": "application/x-www-form-urlencoded",
           "Accept": "text/plain"}
conn = httplib.HTTPConnection("musi-cal.mojam.com:80")
conn.request("POST", "/cgi-bin/query", params, headers)
response = conn.getresponse()
print response.status, response.reason

data = response.read()
conn.close()
```

Request-post

◆带参数头

```
import json
Import requests
url = 'https://api.github.com/some/endpoint'
payload = {'some': 'data'}
headers = {'content-type': 'application/json'}

r = requests.post(url, data=json.dumps(payload), headers=headers)
print r.text
```

Request-post

◆发送文件

```
url = 'http://httpbin.org/post'
files = {'file': ('report.xls', open( 'aa', 'rb'), 'application/vnd.ms-excel', {'Expires': '0'})}
r = requests.post(url, files=files)
print r.text
```

```
POST / HTTP/1.1
Host: 10.110.13.186:8081
Content-Length: 1098
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.6.0 CPython/2.7.5 Linux/3.10.0-327.el7.x86_64
Connection: keep-alive
Content-Type: multipart/form-data; boundary=d25cf3a62ae0461d8876f24070804165

--d25cf3a62ae0461d8876f24070804165
Content-Disposition: form-data; name="file"; filename="report.xls"
Content-Type: application/vnd.ms-excel
Expires: 0

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCHD395SKnRVSVQPXRnCXP2DpUtPcFljprycQ3/9cwIShvIOryA
geDt96WoasnkmTOhMYWq+7zn+rTINoy5EcPQn4qG0aish2dSJS2U/kuw+WcDquCr
```

Pycurl

cURL是一个利用URL语法在命令行下工作的文件传输工具。

它支持文件上传和下载，所以是综合传输工具，但按传统，习惯称cURL为下载工具。

cURL还包含了用于程序开发的libcurl。

cURL支持的通信协议有FTP、FTPS、HTTP、HTTPS、TFTP、SFTP、Gopher、SCP、Telnet、DICT、FILE、LDAP、LDAPS、IMAP、POP3、SMTP和RTSP。

```
curl "www.hotmail.com/when/junk.cgi?birthyear=1905&press=OK"
curl -d "birthyear=1905&press=OK" www.hotmail. com/when/junk.cgi
curl -F upload=@localfilename -F press=OK URL
curl -u name:password www.secrets. com
curl -A "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)" URL
curl -b cookies.txt -c newcookies.txt www.cookiesite. com
```

pycurl

```
# -*- coding: utf-8 -*-
import pycurl
from StringIO import StringIO

buffer = StringIO()
c = pycurl.Curl()
c.setopt(c.URL, 'http://pycurl.io/')
c.setopt(c.WRITEFUNCTION, buffer.write)
c.perform()
c.close()

body = buffer.getvalue()
# Body is a string in some encoding.
# In Python 2, we can print it without knowing what the encoding is.
print(body)
```

Selenium

- Selenium是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，最初是为网站自动化测试而开发的，类型像我们玩游戏用的按键精灵，可以按指定的命令自动化操作，不同是Selenium可以直接运行在浏览器上，它支持所有主流的浏览器(包括PhantomJS这些无界面的浏览器)。
- Selenium可以根据我们的指令，让浏览器自动加载页面，获取需要的页面，甚至页面截屏，或者判断网站上某些动作是否发生。
- Selenium自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。但是我们有时候需要让它内嵌在代码中运行，所有我们而已用一个叫PhantomJS的工具代替真实的浏览器。

Selenium

Selenium库里有一个叫WebDriver的API。WebDriver可以控制浏览器的操作，它可以像BeautifulSoup或者其它Selector对象一样用来查找页面元素，与页面上的元素进行交互(发送文本、点击等)，以及执行其他动作来运行网络爬虫。

```
driver.find_element_by_id('kw').send_keys(u'长城')
driver.find_element_by_id('su').click()
```

定位UI元素(WebElements)

- find_element_by_id
- find_element_by_name
- find_element_by_xpath
- find_element_by_link_text
- find_element_by_partial_link_text
- find_eelement_by_tag_name
- find_element_by_class_name
- find_element_by_css_selector

Selenium支持功能

- 页面操作
- 定位UI元素(WebElements)
- 鼠标动作链
- 填充表单
- 弹窗处理
- 页面切换
- 页面的前进和后退
- Cookies
- 页面等待
  - 显式等待
  - 隐式等待

PhantomJS

- PhantomJS是一个基于Webkit的"无界面"(headless)浏览器，它会把网站加载到内存并执行页面上的JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器更高效。

- 如果我们把Selenium和PhantomJS结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理JavaScript、Cookie、headers，以及任何我们真实用户需要做的事情。

Jinja2

Jinja2 是一个现代的，设计者友好的，仿照 Django 模板的 Python 模板语言。

它速度快，被广泛使用，并且提供了可选的沙箱模板执行环境保证安全:

```
from jinja2 import Template
template = Template('Hello {{ name }}!')
print( template.render(name='John Doe'))
```

flask

Flask是一个Python编写的Web 微框架，让我们可以使用Python语言快速实现一个网站或Web服务.

```
from flask import Flask

app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello Flask!'
if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5001)
```

ftplib

通过ftp协议上传下载文件

ftp相关的命令操作

```
ftp.FTP([host[, user[, passwd[, acct[, timeout]]]]])
ftp.login(user, passwd, acct)
ftp.cwd(pathname)              # 设置FTP当前操作的路径
ftp.dir()                      # 显示目录下所有目录的信息
ftp.nlst()                     # 获取目录下的文件
ftp.mkd(pathname)              # 新建远程目录
ftp.rmd(dirname)               # 删除远程目录
ftp.pwd()                      # 返回当前所在位置
ftp.delete(filename)           # 删除远程文件
ftp.rename(fromname, toname)     #将fromname改为toname
ftp.storbinaly('STOR filename.txt',file_handel,bufsize)  # 上传目标文件
ftp.retrbinary('RETR filename.txt',file_handel,bufsize)  # 下载FTP文件
```

telnetlib

通过telnet协议远程操作服务器

```
import getpass
import sys
import telnetlib
HOST = "localhost"
user = raw_input("Enter your remote account: ")
password = getpass.getpass()
tn = telnetlib.Telnet(HOST)
tn.read_until("login: ")
tn.write(user + "\n")
if password:
    tn.read_until("Password: ")
    tn.write(password + "\n")
tn.write("ls\n")
tn.write("exit\n")
print tn.read_all()
```

paramiko

paramiko模块，基于SSH用于连接远程服务器并执行相关操作。

```
import paramiko
# 创建SSH对象
ssh = paramiko.SSHClient()
# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# 连接服务器
ssh.connect(hostname='c1.salt.com', port=22, username='wupeiqi', password='123')
# 执行命令
stdin, stdout, stderr = ssh.exec_command('ls')
# 获取命令结果
result = stdout.read()
# 关闭连接
ssh.close()
```

Python并发

进程
进程池
线程
协程
锁
递归锁
信号量
条件变量/事件
队列
非阻塞
多路复用

Python并发

```
# -*- coding: utf-8 -*-
import threading
import time
def run(n):
    print("task",n)
    time.sleep(2)

res=[]#存线程实例
start_time=time.time()
for i in range(50):#创建线程50个线程
    t=threading.Thread(target=run,args=("t-%s"%i,))
    t.start()
    res.append(t)
for r in res:#循环线程实例列表，等待所有的线程执行完毕
    r.join()#线程执行完毕后，才会往后执行，相当于C语言中的wait()
print("cost:",time.time()-start_time)#结果是  cost: 2.006114959716797
```

数据缓存

- 文件
- Redis
- Memcache
- Kafka
- Activemq
- Rabbitmq
- zeromq
- CMSP

数据缓存-文件

File 对象和 OS 对象提供了很多文件与目录的操作方法

File 对象方法：file 对象提供了操作文件的一系列方法。

OS 对象方法：提供了处理文件及目录的一系列方法。

http://www.runoob.com/python/python-files-io.html

数据缓存-文件

```
# -*- coding: UTF-8 -*-

import os;
document = open("testfile.txt", "w+");
print "文件名: ", document.name;
document.write("这是我创建的第一个测试文件！\nwelcome!");
print document.tell();
#输出当前指针位置
document.seek(os.SEEK_SET);
#设置指针回到文件最初
context = document.read();
print context;
document.close();
```

数据缓存-memcache

memcache是一套分布式的高速缓存系统，尤其对于一些大型的、需要频繁访问数据库的网站访问速度提升效果十分显著。

MemCache的工作流程如下：先检查客户端的请求数据是否在memcached中，如有，直接把请求数据返回，不再对数据库进行任何操作；如果请求的数据不在memcached中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcached中（memcached客户端不负责，需要程序明确实现）；每次更新数据库的同时更新memcached中的数据，保证一致性；当分配给memcached内存空间用完之后，会使用LRU（Least Recently Used，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

Memcache是一个高性能的分布式的内存对象缓存系统，通过在内存里维护一个统一的巨大的hash表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。简单的说就是将数据调用到内存中，然后从内存中读取，从而大大提高读取速度。

Memcache是danga的一个项目，最早是LiveJournal 服务的，最初为了加速 LiveJournal 访问速度而开发的，后来被很多大型的网站采用。

Memcached是以守护程序(监听)方式运行于一个或多个服务器中，随时会接收客户端的连接和操作
https://pypi.org/project/python-memcached/#files

数据缓存-memcache

```
#!/usr/bin/env python
import memcache
mc = memcache.Client(['127.0.0.1:12000'],debug=0)
mc.set("some_key", "Some value")
value = mc.get("some_key")
mc.set("another_key", 3)
mc.delete("another_key")

mc.set("key", "1")   # note that the key used for incr/decr
must be a string.
mc.incr("key")
mc.decr("key")
```

数据缓存-redis

Redis是一个开源的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

它通常被称为数据结构服务器，类型包括，

字符串(String)，

哈希(Map)，

列表(list)，

集合(sets)

有序集合(sorted sets)等类型。

http://www.runoob.com/redis/redis-tutorial.html

数据缓存-redis

```
import redis

pool = redis.ConnectionPool(host='192.168.1.150',
port=6379)
r = redis.Redis(connection_pool=pool)
r.set('foo', 'Bar')
print(r.get('foo'))
```

数据缓存-kafka

Kafka 是一种高吞吐量 的分布式发布订阅消息系统，有如下特性：

通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。

高吞吐量 ：即使是非常普通的硬件Kafka也可以支持每秒数百万 的消息。

支持通过Kafka服务器和消费机集群来分区消息。

支持Hadoop并行数据加载。

数据缓存-kafka

```
pip install kafka-python
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers='localhost:1234')
for _ in range(100):
        producer.send('foobar', b'some_message_bytes')


consumer =
KafkaConsumer('test',bootstrap_servers=['172.21.10.136:9092'])
for message in consumer:
    print ("%s:%d:%d: key=%s value=%s" % (message.topic,
message.partition, message.offset, message.key, message.value))
```

数据缓存-activemq

ActiveMQ是消息中间件，是一种在分布式系统中应用程序借以传递消息的媒介，常用的有ActiveMQ，RabbitMQ，kafka。ActiveMQ是Apache下的开源项目，完全支持JMS1.1和J2EE1.4规范的JMS Provider实现。

特点：

1、支持多种语言编写客户端

2、对spring的支持，很容易和spring整合

3、支持多种传输协议：TCP,SSL,NIO,UDP等

4、支持AJAX

消息形式：

1、点对点（queue）

2、一对多（topic）

数据缓存-activemq

```
import time
import sys
import stomp
class MyListener(object):
    def on_error(self, headers, message):
        print('received an error %s' % message)
    def on_message(self, headers, message):
        print('received a message %s' % message)
#官方示例的连接代码也落后了，现在分协议版本
conn = stomp.Connection10([('ip...',61613)])
conn.set_listener('', MyListener())
conn.start()
conn.connect()
conn.subscribe(destination='/queue/test', id=1, ack='auto')
#注意，官方示例这样发送消息是有问题的
#conn.send(body='hello,garfield! this is '.join(sys.argv[1:]), destination='/queue/test')
conn.send(body='hello,garfield!', destination='/queue/test')
time.sleep(2)
conn.disconnect()
```

数据缓存-rabbitmq

RabbitMQ是一个在AMQP基础上完成的，可复用的企业消息系统。他遵循Mozilla Public License开源协议。

RabbitMQ 概念介绍：

Message 消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的

Publisher 消息的生产者

Exchange 交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。

Binding 绑定，用于消息队列和交换器之间的关联。

Connection 网络连接，比如一个TCP连接。

Channel 信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内地虚拟连接

Consumer 消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

Virtual Host 虚拟主机，表示一批交换器、消息队列和相关对象。

Broker 表示消息队列服务器实体

数据缓存-rabbitmq

```
#!/usr/bin/env python
import pika
connection =
pika.BlockingConnection(pika.ConnectionParameters(host='localhost'
))
channel = connection.channel()
channel.queue_declare(queue='hello')

channel.basic_publish(exchange='',
                      routing_key='hello',
                      body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

数据缓存-zeromq

ZeroMQ是一种基于消息队列的多线程网络库，其对套接字类型、连接处理、帧、甚至路由的底层细节进行抽象，提供跨越多种传输协议的套接字。ZeroMQ是网络通信中新的一层，介于应用层和传输层之间（按照TCP/IP划分），其是一个可伸缩层，可并行运行，分散在分布式系统间.

ZeroMQ几乎所有的I/O操作都是异步的，主线程不会被阻塞。ZeroMQ会根据用户调用zmq_init函数时传入的接口参数，创建对应数量的I/O Thread。每个I/O Thread都有与之绑定的Poller，Poller采用经典的Reactor模式实现，Poller根据不同操作系统平台使用不同的网络I/O模型（select、poll、epoll、devpoll、kequeue等）。

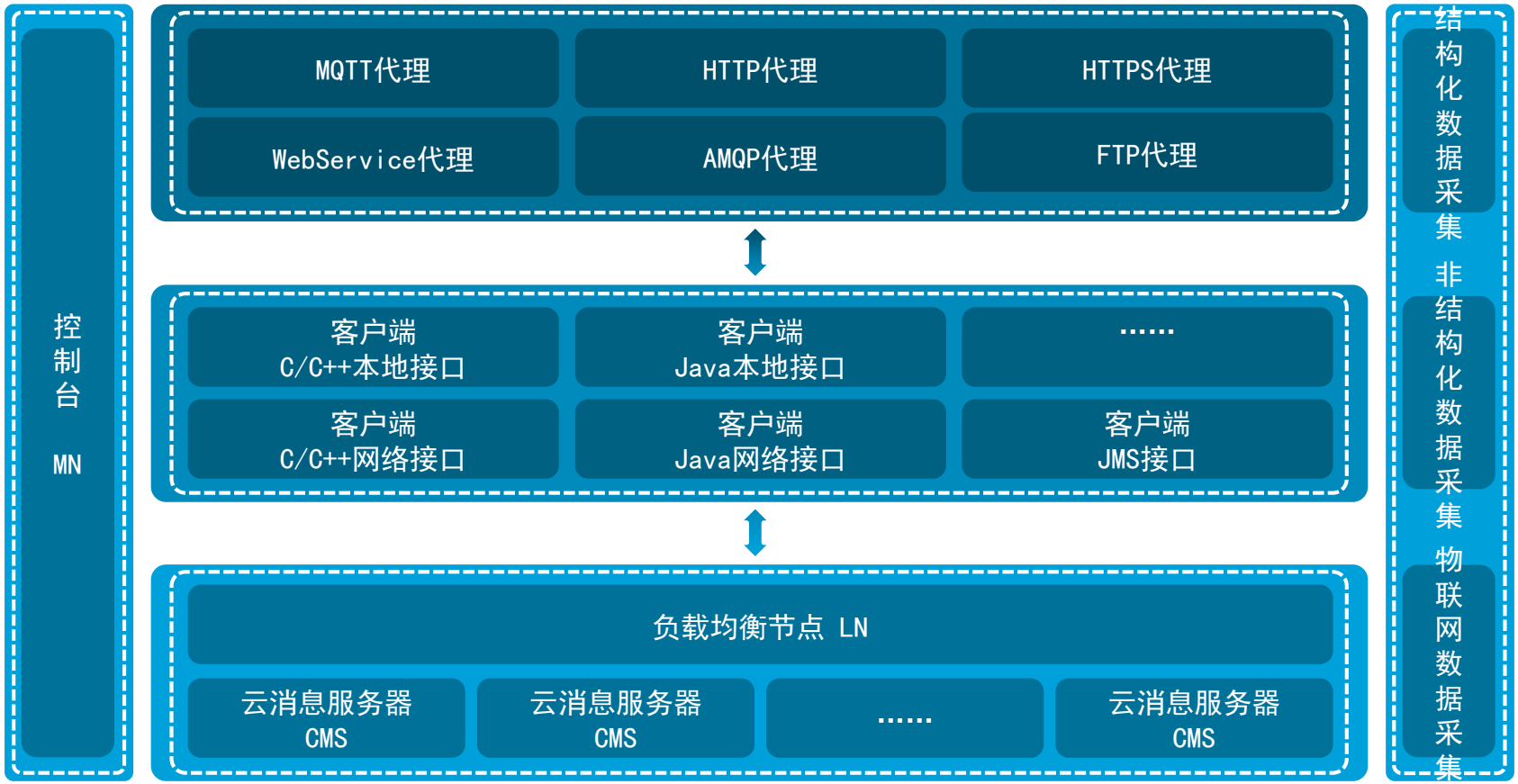
数据缓存-zeromq

```
#!/usr/bin/python
#-*-coding:utf-8-*-
import zmq
import sys
context = zmq.Context()
socket = context.socket(zmq.REQ)
socket.connect("tcp://localhost:5555")
while(True):
    data = raw_input("input your data:")
    if data == 'q':
        sys.exit()
    socket.send(data)
    response = socket.recv();
    print response
```

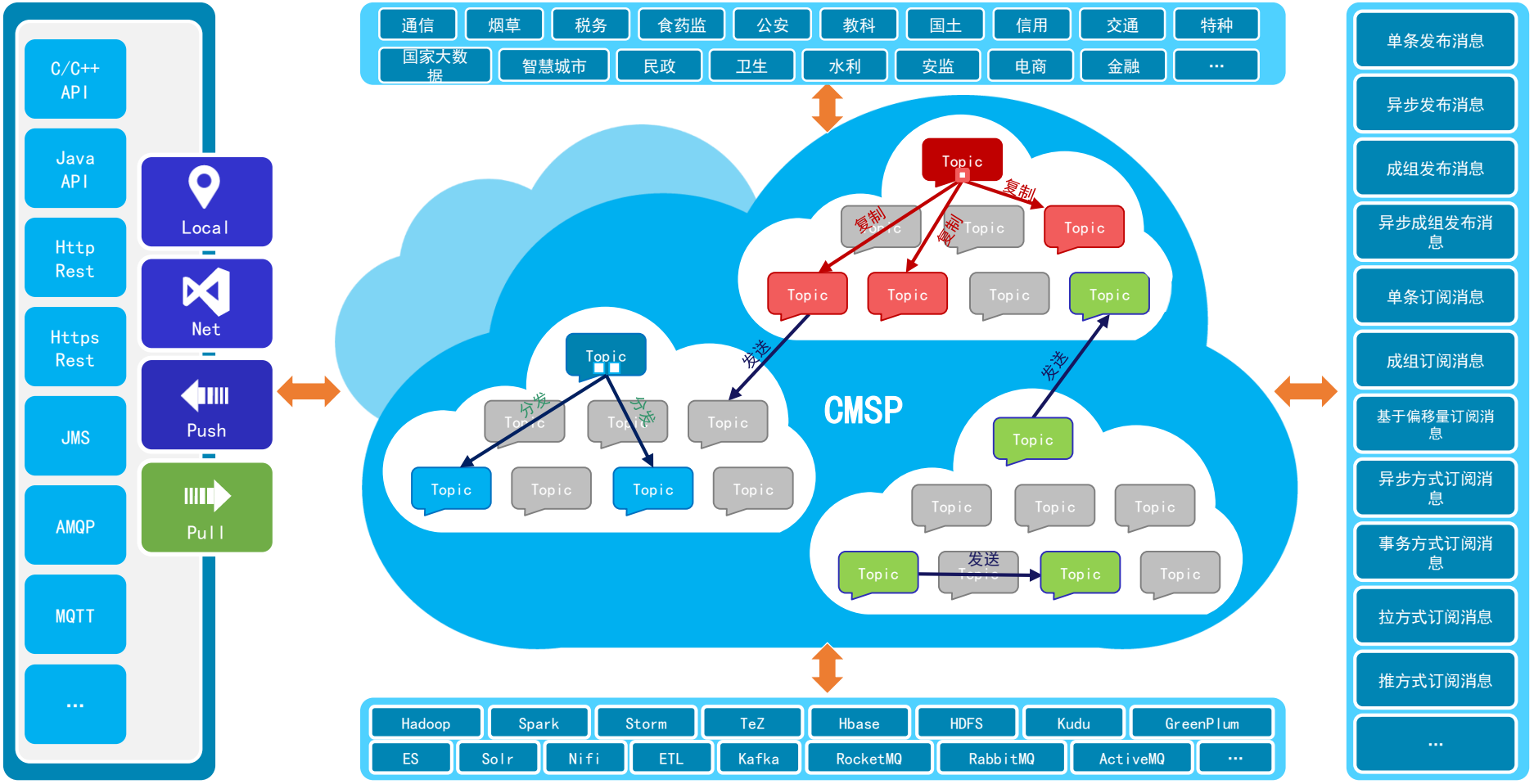
CMSP

- CMSP是云和大数据时代的一种高性能消息中间件，以云服务和集群方式对外提供高性能和高可靠的消息队列服务，支撑大数据采集、传输、汇聚、交换，海量数据实时处理和微服务处理架构
- CMSP是一种高效、可靠、安全、便捷、可弹性扩展的分布式消息服务系统，它的目标是高可靠地在一个云消息服务引擎上可以支持成千上万的应用7*24不间断并发访问，在廉价PC服务器集群上支持每秒上亿条的消息存取服务，支持随业务量需要动态增加或减少计算资源节点，同时要确保基于云消息服务平台的应用开发和管理简单易使用

产品架构


控制台
MN
MQTT代理
HTTP代理
HTTPS代理
WebService代理
AMQP代理
FTP代理
客户端
C/C++本地接口
客户端
Java本地接口
.....
客户端
C/C++网络接口
客户端
Java网络接口
客户端
JMS接口
负载均衡节点 LN
云消息服务器
CMS
云消息服务器
CMS
.....
云消息服务器
CMS
结构化数据采集
非结构化数据采集
物联网数据采集

产品功能介绍


C/C++ API
Java API
Http Rest
Https Rest
JMS
AMQP
MQTT
...
Local
Net
Push
Pull
通信
烟草
税务
食药监
公安
教科
国土
信用
交通
特种
国家大数据
智慧城市
民政
卫生
水利
安监
电商
金融
...
CMSP
Topic
复制
发送
分发
Hadoop
Spark
Storm
TeZ
Hbase
HDFS
Kudu
GreenPlum
ES
Solr
Nifi
ETL
Kafka
RocketMQ
RabbitMQ
ActiveMQ
...
单条发布消息
异步发布消息
成组发布消息
异步成组发布消息
单条订阅消息
成组订阅消息
基于偏移量订阅消息
异步方式订阅消息
事务方式订阅消息
拉方式订阅消息
推方式订阅消息
...

CMSP

```
#!/usr/bin/python
from string import atoi
from time import sleep
from sys import *
from cmspcli import *
if __name__ == "__main__":
    ip="10.110.13.101"
    port=1201
    user="imqAdmin"
    passwd="pw2017@GL!"
    topic=""
    cloud=0
    cl = CmspCli()
    cl.connectcmsp(ip,port,usr,passwd,topic,cloud)
    topiclist=cl.getalltopic()
    for topic in topiclist:
        print  topic

cl.opentopic("t1")
infolist = cl.gettopicinfo("t1")
for line in infolist:
    print line
print cl.getqueuefilename("t2")
print "t2 count = ", cl.getmsgcount("t2")
cl.putmq("aaaaaaaaaaaaa")
print "t2 count = ", cl.getmsgcount("t2")

cl.closecmsp()
```

数据持久化

数据持久化

- Csv
- ini
- Xml
- Json
- Excel
- Sqlite
- Mysql
- Mongodb
- Hdfs
- Hbase

数据存储-csv

```
import csv
with open('a.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
    for row in spamreader:
        print(' | '.join(row))

import csv
with open('eggs.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=' ',
                            quotechar='|',
quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(['Spam'] * 5 + ['Baked Beans'])
    spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
```

数据存储-ini

```
import configparser
config = configparser.ConfigParser()
config['DEFAULT'] = {'ServerAliveInterval': '45',
                     'Compression': 'yes',
                     'CompressionLevel': '9'}
config['bitbucket.org'] = {}
config['bitbucket.org']['User'] = 'hg'
config['topsecret.server.com'] = {}
topsecret = config['topsecret.server.com']
topsecret['Port'] = '50022'      # mutates the parser
topsecret['ForwardX11'] = 'no'   # same here
config['DEFAULT']['ForwardX11'] = 'yes'
with open('example.ini', 'w') as configfile:
    config.write(configfile)
```

数据存储-ini

```
import configparser
config = configparser.ConfigParser()
config.read('example.ini')
config.sections()
config['bitbucket.org']['User']
config['DEFAULT']['Compression']
topsecret = config['topsecret.server.com']
topsecret['ForwardX11']
topsecret['Port']
for key in config['bitbucket.org']:
    print(key)
```

数据存储-读取配置文件（ini，conf）

```
import configparser
config = configparser.ConfigParser()
config.read('example.ini')

print(config.get('bitbucket.org', 'User'))
#config.remove_option('Section1', 'baz')
print(config.get('Section1', 'foo',fallback="default
value"))
```

数据存储-xml

- python对XML的解析
- 常见的XML编程接口有DOM和SAX，这两种接口处理XML文件的方式不同，当然使用场合也不同。
- python有三种方法解析XML，SAX，DOM，以及ElementTree:
- 1.SAX (simple API for XML)

  python 标准库包含SAX解析器，SAX用事件驱动模型，通过在解析XML的过程中触发一个个的事件并调用用户定义的回调函数来处理XML文件。
- 2.DOM(Document Object Model)

  将XML数据在内存中解析成一个树，通过对树的操作来操作XML。
- 3.ElementTree(元素树)

  ElementTree就像一个轻量级的DOM，具有方便友好的API。代码可用性好，速度快，消耗内存少。

注：因DOM需要将XML数据映射到内存中的树，一是比较慢，二是比较耗内存，而SAX流式读取XML文件，比较快，占用内存少，但需要用户实现回调函数（handler）。

数据解析-xml parse

```
from xml.etree.ElementTree import XMLParser
import xml.etree.ElementTree as ET

tree = ET.parse('movies.xml')
root = tree.getroot()

exampleXml=ET.tostring(root)
parser = ET.XMLPullParser(['start', 'end'])
parser.feed(exampleXml)
#print(parser.read_events())
for event, elem in parser.read_events():
    print(event)
    print(elem.tag, 'text=', elem.text)
```

数据存储-json

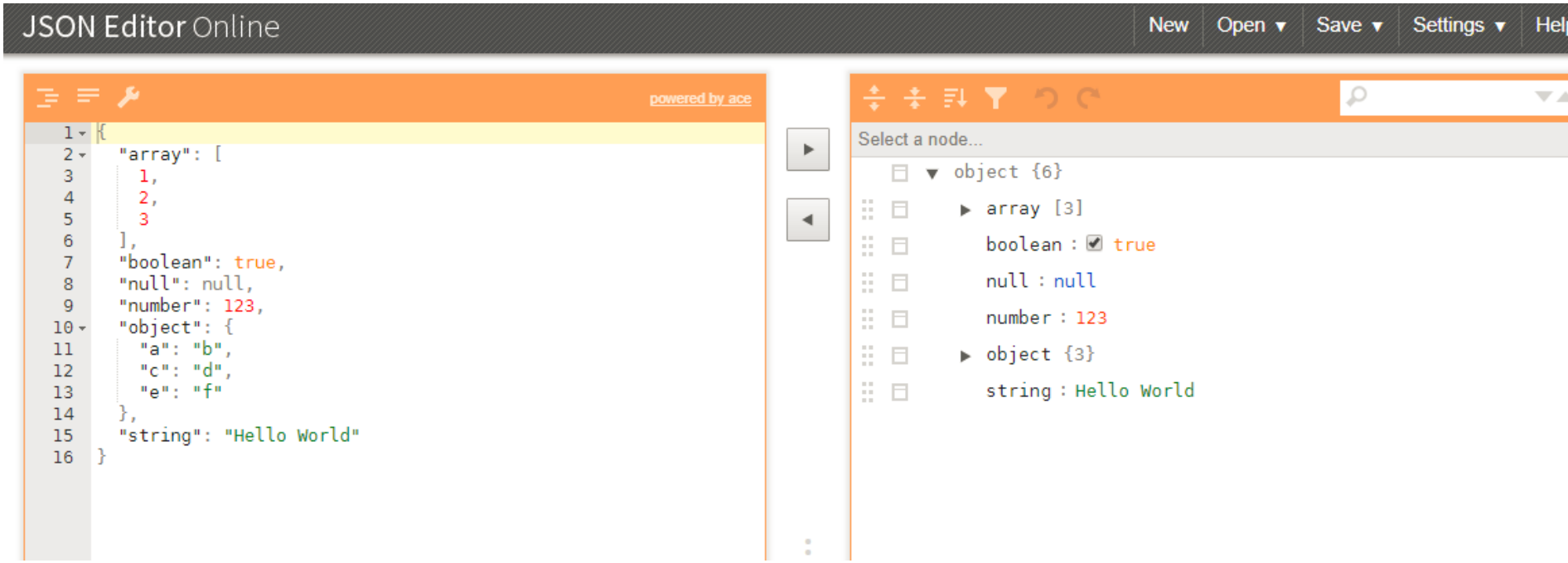
JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false
None	null

http://jsoneditoronline.org/

数据存储-json在线检验

http://jsoneditoronline.org/



数据存储-json在线检验

http://zaa.ch/jsonlint/

https://www.jsonschemavalidator.net/

数据存储-json

```
import json
data = {   'name' : 'ACME',    'shares' : 100}
json_str = json.dumps(data)   # 编码
data = json.loads(json_str)   # 解码
// load 可以从字符串，STRINGIO，文件种加载
```

还有第三方demjson

demjson 是 python 的第三方模块库，可用于编码和解码 JSON 数据，包含了 JSONLint 的格式化及校验功能。

数据存储-对象序列化

- 用于序列化的两个模块
  - json：用于字符串和Python数据类型间进行转换
  - pickle：用于python特有的类型和python的数据类型间进行转换
  - json提供四个功能：dumps,dump,loads,load
  - pickle提供四个功能：dumps,dump,loads,load
- **pickle可以存储什么类型的数据呢？**
- 所有python支持的原生类型：布尔值，整数，浮点数，复数，字符串，字节，None。
- 由任何原生类型组成的列表，元组，字典和集合。
- 函数，类，类的实例

数据存储-excel

Xlrd excel xls 读取

https://pypi.org/project/xlrd/#files

Xlwt excel xls 写入

https://pypi.org/project/xlwt/#files

XlsxWriter xlsx富文本操作

https://pypi.org/project/XlsxWriter/#files

http://xlsxwriter.readthedocs.io/

http://xlsxwriter.readthedocs.io/example_demo.html#ex-demo

数据存储-excel

```
# -*- coding: utf-8 -*-
import xlrd
xlsfile = r'e:\test1.xls'
book = xlrd.open_workbook(xlsfile)
sheet0 = book.sheet_by_index(0)
print "1 -",sheet0
sheet_name = book.sheet_names()[0]
print "2 -",sheet_name
sheet1 = book.sheet_by_name(sheet_name)

#循环打印每一行的内容
for i in range(nrows):
    for j in range(ncols):
        cell_value = sheet0.cell_value(i,j)
        print cell_value,"\t",
    print
```

数据存储-sqlite

SQLite 是一个软件库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。SQLite 是在世界上最广泛部署的 SQL 数据库引擎。SQLite 源代码不受版权限制。

您不需要单独安装该模块，因为 Python 2.5.x 以上版本默认自带了该模块。

http://www.runoob.com/sqlite/sqlite-tutorial.html

http://www.runoob.com/sqlite/sqlite-python.html

数据存储-sqlite

```
#!/usr/bin/python
import sqlite3
conn = sqlite3.connect('test.db')
print "Opened database successfully";
c = conn.cursor()
c.execute('''CREATE TABLE COMPANY
       (ID INT PRIMARY KEY     NOT NULL,
       NAME           TEXT    NOT NULL,
       AGE            INT     NOT NULL,
       ADDRESS        CHAR(50),
       SALARY         REAL);''')
print "Table created successfully";
conn.commit()
conn.close()
```

数据存储-mysql

MySQL数据库

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

- MySQL 是开源的，所以你不需要支付额外的费用。
- MySQL 支持大型的数据库。可以处理拥有上千万条记录的大型数据库。
- MySQL 使用标准的SQL数据语言形式。
- MySQL 可以运行于多个系统上，并且支持多种语言。这些编程语言包括C、C++、Python、Java、Perl、PHP、Eiffel、Ruby和Tcl等。
- MySQL 对PHP有很好的支持，PHP是目前最流行的Web开发语言。
- MySQL 支持大型数据库，支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB，64位系统支持最大的表文件为8TB。
- MySQL 是可以定制的，采用了GPL协议，你可以修改源码来开发自己的 MySQL 系统。

http://www.runoob.com/mysql/mysql-tutorial.html
http://www.runoob.com/python/python-mysql.html

数据存储-mongodb

什么是MongoDB？

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。

在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

```
{
   name: "sue",                          <-- field: value
   age: 26,                              <-- field: value
   status: "A",                          <-- field: value
   groups: [ "news", "sports" ]          <-- field: value
}
```

http://www.runoob.com/mongodb/mongodb-intro.html
http://www.runoob.com/python3/python-mongodb.html

数据存储-mongodb

```
#!/usr/bin/python3
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["runoobdb"]
mycol = mydb["sites"]
mydict = { "name": "RUNOOB", "alexa": "10000", "url":
"https://www.runoob.com" }
x = mycol.insert_one(mydict)

print(x)
```

数据存储-hdfs

Hadoop分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS放宽了一部分POSIX约束，来实现流式读取文件系统数据的目的。HDFS在最开始是作为Apache Nutch搜索引擎项目的基础架构而开发的。HDFS是Apache Hadoop Core项目的一部分。

数据存储-hdfs

```
pip install hdfs
client = Client("http://127.0.0.1:50070",root="/",timeout=100,session=False)
client.list("/")
client.status("/")
client.makedirs("/test",permission=777)
client.rename("/test","/new_name")
client.delete("/new_name")
client.upload("/test","/opt/bigdata/hadoop/NOTICE.txt")
client.download("/test/NOTICE.txt","/home")
with client.read("/test/NOTICE.txt") as reader
    print reader.read()
```

数据存储-hbase

HBase是建立在Hadoop文件系统之上的分布式面向列的数据库。它是一个开源项目，是横向扩展的。

HBase是一个数据模型，类似于谷歌的大表设计，可以提供快速随机访问海量结构化数据。它利用了Hadoop的文件系统（HDFS）提供的容错能力。

它是Hadoop的生态系统，提供对数据的随机实时读/写访问，是Hadoop文件系统的一部分。

人们可以直接或通过HBase的存储HDFS数据。使用HBase在HDFS读取消费/随机访问数据。 HBase在Hadoop的文件系统之上，并提供了读写访问。

数据存储-hbase

```
pip3 install hbase-python
```

```
import hbase
zk = 'sis3.ustcdm.org:2181,sis4.ustcdm.org:2181'
if __name__ == '__main__':
    with hbase.ConnectionPool(zk).connect() as conn:
        table = conn['mytest']['videos']
        row = table.get('00001')
        print(row)
    exit()
```