XML解析

## 理解

标记语言，是一种将文本（ Text）以及文本相关的其他信息结合起来，展现出关

于文档结构和数据处理细节的电脑文字编码。当今广泛使用的标记语言是超文本

标记语言（HyperText Markup Language， HTML）和可扩展标记语言 (Extensible

Markup Language XML)。标记语言广泛应用于网页和网络应用程序。

### 超文本标记语言HTML

a. 写法格式： <a href="link.html">link</a>

b. 关注数据的展示与用户体验

c. 标记是预定义、不可扩展的（如 <a></a>表示超链接）

### 可扩展标记语言XML

1. 写法格式：同 html 样式
2. 仅关注数据本身
3. 标记定义

xml 和 Html 语言由同一种父语言 SGML(Standard Generalized Markup

Language, 标准通用标记语言)发展出来的两种语言。

xml 由 html 发展而来，与 html 格式相似，但是比 html 严格。 XML描述的是结构、内容和语义，它不描述页面元素的格式化。 HTML侧重于如何表现信息，内容描述与显示整合为一体。 XML中的每个元素名都是成对出现的。结束标签前加一个/如:<foot></foot>

## 作用

1. 存储、交换数据
2. 配置

## 使用

### 1、方式一

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?><!-- XML声明必须写在首行，固定格式 --> <!-- 注释：根节点(元素)people -->* <**people**>  *<!-- 子节点(元素)person 属性name,age -->* <**person**>  <**name**>王晶晶</**name**>  <**age**>28</**age**>  </**person**>   <**person**>  <**name**>至尊宝</**name**>  <**age**>300</**age**>  </**person**> </**people**> |

### 2、方式二

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?><!-- XML声明必须写在首行，固定格式 --> <!-- 注释：根节点(元素)people -->* <**people**>  *<!-- 子节点(元素)person 属性name,age -->* <**person name="王晶晶" age="28"**></**person**>  <**person name="至尊宝" age="300"**></**person**> </**people**> |

## 语法

### 语法规范

1. 必须要有XML声明，首行编写
2. 必须有且只有一个根元素
3. 严格区分大小写
4. 属性值用引号(双引号或单引号)，如：属性名=**"**属性值**"**；在一个元素上，相同的属性只能出现一次
5. 标记成对编写
6. 空标记关闭，如：</br>
7. 元素正确嵌套(格式问题，成对编写)

### 元素命名规则

a. 名称中可以包含字母、数字或者其他可见字符；

b. 名称不能以数字开头；

c. 不能以XML/xml/Xml…开头(关键字)；

d. 名称中不能含空格；

e. 名称中不能含冒号（注：冒号留给命名空间使用）

### 3、 实体：&实体名;

在xml元素中，< 和 & 是非法的，

< 会产生错误，因为解析器会把该字符解释为新元素的的开始；

& 会产生错误，因为解析器会把该字符解释为字符实体的开始，

把它们替换为实体引用是个好的习惯。

#### 3.1、内置的5种实体

实体 符号

&lt; <

&gt; >

&amp; &

&quot; "

&apos; '

#### 3.2、自定义实体

<!DOCTYPE 根元素[

<!--实体声明-->

<!ENTITY 实体名 "实体内容">

]>

#### 3.3、使用自定义实体

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE sxt**[ <!-- 实体声明 --> <!ENTITY xm "张三"> <!ENTITY gen "保密"> ]***>*** <**sxt**>  <**teacher**>  <**name**>张三</**name**>  <**gender**>保密</**gender**>  </**teacher**>  *<!-- 引用实体的方法 -->* <**teacher**>  <**name**>**&xm;**</**name**>  <**gender**>**&gen;**</**gender**>  </**teacher**> </**sxt**> |

#### 3.4、文档类型声明

文档类型定义——DOCTYPE，文档类型声明，紧跟XML声明之后，包含所有实体声明

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE 根元素标记名[

<!--实体声明-->

]>

## 组成

### 构成

所有的 XML 文档（以及 HTML 文档）均由以下简单的构建模块构成：

1. 元素
2. 属性
3. 实体
4. PCDATA
5. CDATA
6. CDATA节（character data）

### CDATA(原生字符)

CDATA部分中的所有内容都会被解析器忽略

用于把整段文本解释为纯字符数据而不是标记的情况。如包含大量的<、>、&、或者"字符。CDATA节中的所有字符都会被当做元素字符数据的常量部分，而不是XML标记，可以输入除]]>外任意字符，不能嵌套。

格式：<![CDATA[ ...... ]]>

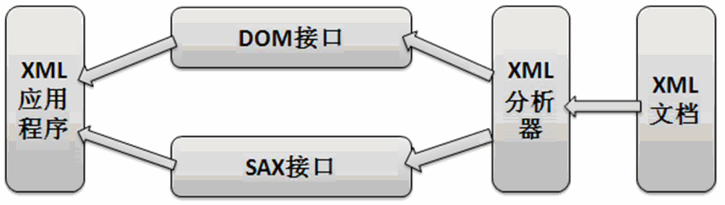
### PCDATA

PCDATA部分中的所有内容都会被解析器执行，所以需要使用实体引用

PCDATA 的意思是被解析的字符数据（parsed character data）。可把字符数据想象为 XML 元素的开始标签与结束标签之间的文本。PCDATA 是会被解析器解析的文本。这些文本将被解析器检查实体以及标记。文本中的标签会被当作标记来处理，而实体会被展开。不过，被解析的字符数据不应当包含任何 & < > 字符；需要使用 &amp; &lt; &gt; 实体来分别替换它们。

## XML解析

基本的解析方式有两种:一种叫SAX，另一种叫DOM。SAX(Simple API for XML)是基于事件流的解析，DOM(Document Object Model)是基于XML文档树结构的解析。SAX:效率高，数据量小，仅一次获取，DOM:整颗树加载到内存中，耗内存，可多次获取。



### SAX解析

SAX(Simple API for XML)解析是按照xml文件的顺序一步一步的来解析。SAX没有官方的标准机构，它不属于任何标准组织或团体，也不属于任何公司或个人，而是提供任何人使用的一种计算机技术。

SAX(Simple API for XML,操作XML的简单接口)，与DOM操作不同的是，SAX采用的是一种顺序的模式进行访问，是一种快速读取XML数据的方式。当使用SAX解析器进行操作的时候会触发一系列的事情，当扫描到文档(document)开始与结束、元素（element）开始与结束时都会调用相关的处理方法，并由这些操作方法作出相应的操作，直至整个文档扫描结束。如果要想实现这种SAX解析，则肯定首先建立一个SAX的解析器。

优点就是边读边解析，性能高，但不能重复解析

解析流程：

1. 创建一个XMLReader：parser = xml.sax.make\_parser()

parser.setFeature(xml.sax.handler.feature\_namespaces, 0)关闭命名空间解析

b. 编写一个类继承xml.sax.ContentHandler类，并重写startElement（开始），endElement（结束）和characters（接收xml内容）

c. 实例化解析对象：parser.setContentHandler(解析对象)

d. 调用解析方法：parser.parse("文档.xml")

#### 方式一

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.sax  *# 2.构建解析对象继承xml.sax.ContentHandler* **class** MyHandler(xml.sax.ContentHandler):   *# 初始化实例属性* **def** \_\_init\_\_(self):  self.people = **''** self.person = **''** self.name = **''** self.age = 0  self.current\_tag = **''** *# 当前节点**# 开始解析节点执行的方法* **def** startElement(self, tag\_name, tag\_attr):  *# print("解析开始----------------start->", tag\_name, tag\_attr)  # 处理name和age节点* **if 'name'** == tag\_name:  *# print(tag\_name)* self.current\_tag = tag\_name   **if 'age'** == tag\_name:  *# print(tag\_name)* self.current\_tag = tag\_name   *# 结束解析节点执行的方法* **def** endElement(self, tag\_name):  *# print("解析结束----------------end->", tag\_name)  # 解析结束时做统一打印处理* **if** self.current\_tag == **"name"**:  print(**"Person Name:"**, self.name)  **elif** self.current\_tag == **"age"**:  print(**"Person Age:"**, self.age)   self.current\_tag = **""** *# 读取字符执行的方法* **def** characters(self, content):  *# print("解析中-读取字符->'", content, "'")* **if 'name'** == self.current\_tag:  *# print(content)* self.name = content   **if 'age'** == self.current\_tag:  *# print(content)* self.age = content   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  *# 1.创建一个XMLReader* parser = xml.sax.make\_parser()  *# 关闭xml命名空间解析* parser.setFeature(xml.sax.handler.feature\_namespaces, 0)  *# 3.实例化解析对象* handler = MyHandler()  parser.setContentHandler(handler)  *# 4.解析xml文件* parser.parse(**"people.xml"**) |
| *people.xml：*  *<?***xml version="1.0" encoding="utf-8"***?><!-- XML声明必须写在首行且无空格，固定格式 --> <!-- 注释：根节点(元素)people -->* <**people**>  *<!-- 子节点(元素)person 属性name,age -->* <**person**>  <**name**>王晶晶</**name**>  <**age**>28</**age**>  </**person**>   <**person**>  <**name**>至尊宝</**name**>  <**age**>300</**age**>  </**person**> </**people**> |
| *控制台：*  *Person Name: 王晶晶*  *Person Age: 28*  *Person Name: 至尊宝*  *Person Age: 300* |

#### 方式二

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.sax  **class** Person(object):   **def** \_\_init\_\_(self, name, age):  self.name = name  self.age = age   **def** \_\_str\_\_(self):  **return 'Person Name:{self.name}, Person Age:{self.age}'**.format(self = self)   *# 2.构建解析对象继承xml.sax.ContentHandler* **class** MyHandler(xml.sax.ContentHandler):   *# 初始化实例属性* **def** \_\_init\_\_(self):  self.persons = [] *# person列表   # 开始解析节点执行的方法* **def** startElement(self, tag\_name, tag\_attr):  *# print("解析开始----------------start->", tag\_name, tag\_attr)  # 处理name和age属性* **if 'person'** == tag\_name:  *# print(tag\_name)  # print(tag\_attr['name'], tag\_attr['age'])  # 添加至列表中* self.persons.append(Person(tag\_attr[**'name'**], tag\_attr[**'age'**]))   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  *# 1.创建一个XMLReader* parser = xml.sax.make\_parser()  *# 关闭xml命名空间解析* parser.setFeature(xml.sax.handler.feature\_namespaces, 0)  *# 3.实例化解析对象* handler = MyHandler()  parser.setContentHandler(handler)  *# 4.解析xml文件* parser.parse(**"people2.xml"**)   **for** i **in** handler.persons:  print(i) |
| *people.xml:*  *<?***xml version="1.0" encoding="utf-8"***?><!-- XML声明必须写在首行且无空格，固定格式 --> <!-- 注释：根节点(元素)people -->* <**people**>  *<!-- 子节点(元素)person 属性name,age -->* <**person name="王晶晶" age="28"**></**person**>  <**person name="至尊宝" age="300"**></**person**> </**people**> |
| *控制台：*  *Person Name:王晶晶, Person Age:28*  *Person Name:至尊宝, Person Age:300* |

### DOM解析

DOM(Document Object Model， 文档对象模型)，在应用程序中，基于DOM的XML分析器将一个XML文档转换成一个对象模型的集合（通常称为DOM树），应用程序正是通过对这个对象模型的操作，来实现对XML文档数据的操作。

XML本身是以树状的形式出现的，所以DOM操作的时候，也将按照树的形式进行转换。

在整个DOM树种，最大的地方指的是Document，表示一个文档，在这个文档中存在一个根节点。文档下面有Element叫元素，元素页面有Node叫节点

注意：在使用DOM操作的时候，每一个文字的区域也是一个节点， 称为文本节点叫Node。

#### 2.1、创建xml

创建流程：

1. 在内存中创建一个空的文档：doc = xml.dom.minidom.Document()
2. 创建一个根节点：root = doc.createElement('节点名')

设置根节点的属性

1. 将根节点添加到文档对象中：doc.appendChild(root)
2. 创建根节点的子节点并添加到根节点：root.appendChild(子节点)
3. 写出xml文档：

doc.writexml(文件, indent="\n", addindent="\t", encoding='utf-8')

tips: 所有的节点创建都是文档创建，所有的节点添加都是appendChild()

##### 方式一

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.dom.minidom  *# 在内存中创建一个空的文档* doc = xml.dom.minidom.Document()  *# 创建一个根节点people* root = doc.createElement(**'people'**)  *# 设置根节点的属性 id='people'* root.setAttribute(**'id'**, **'people'**)  *# 将根节点添加到文档对象中* doc.appendChild(root)  *# 创建people的子节点 # 准备数据* data = [{**'name'**:**'白晶晶'**, **'age'**:28}, {**'name'**:**'至尊宝'**, **'age'**:300}]  *# 设置属性* **for** i **in** data:  *# 创建person节点* node\_person = doc.createElement(**'person'**)  *# 创建name节点* node\_name = doc.createElement(**'name'**)  *# name节点添加文本节点* node\_name.appendChild(doc.createTextNode(i[**'name'**]))  *# 创建age节点* node\_age = doc.createElement(**"age"**)  *# age节点添加文本节点* node\_age.appendChild(doc.createTextNode(str(i[**"age"**])))   *# 将子节点添加到父节点person中* node\_person.appendChild(node\_name)  node\_person.appendChild(node\_age)   *# 将person节点添加到根节点people中* root.appendChild(node\_person)   *# 写出xml文件* **with** open(**'people\_create.xml'**, **'w'**, encoding=**'utf-8'**) **as** f:  doc.writexml(f, indent=**"\n"**, addindent=**"\t"**, encoding=**'utf-8'**) |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**people id="people"**>  <**person**>  <**name**>白晶晶</**name**>  <**age**>28</**age**>  </**person**>  <**person**>  <**name**>至尊宝</**name**>  <**age**>300</**age**>  </**person**> </**people**> |

##### 方式二

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.dom.minidom  *# 在内存中创建一个空的文档* doc = xml.dom.minidom.Document()  *# 创建一个根节点people* root = doc.createElement(**'people'**)  *# 设置根节点的属性 id='people'* root.setAttribute(**'id'**, **'people'**)  *# 将根节点添加到文档对象中* doc.appendChild(root)  *# 创建people的子节点 # 准备数据* data = [{**'name'**:**'白晶晶'**, **'age'**:28}, {**'name'**:**'至尊宝'**, **'age'**:300}]  *# 设置属性* **for** i **in** data:  *# 创建person节点* node\_person = doc.createElement(**'person'**)  *# 设置name属性* node\_person.setAttribute(**'name'**, i[**'name'**])  *# 设置age属性* node\_person.setAttribute(**'age'**, str(i[**'age'**]))   *# 将person节点添加到根节点people中* root.appendChild(node\_person)   *# 写出xml文件* **with** open(**'people\_create2.xml'**, **'w'**, encoding=**'utf-8'**) **as** f:  doc.writexml(f, indent=**"\n"**, addindent=**"\t"**, encoding=**'utf-8'**) |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**people id="people"**>  <**person age="28" name="白晶晶"**/>  <**person age="300" name="至尊宝"**/> </**people**> |

dom对象树生成以后，可以调用dom的writexml()方法来将内容写入文件中。writexml()方法语法格式为：

writexml(writer, indent, addindent, newl, encoding)

writer是文件对象

indent是每个tag前填充的字符，如：’ ‘，则表示每个tag前有两个空格

addindent是每个子结点的缩近字符

newl是每个tag后填充的字符，如：’\n’，则表示每个tag后面有一个回车

encoding是生成的XML信息头中的encoding属性值，在输出时minidom并不真正进行编码的处理，如果你保存的文本内容中有汉字，则需要自已进行编码转换。

writexml方法writer参数必须要有，其余可以省略。

#### 2.2、解析xml

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.dom.minidom  *# 读取xml文件* dom\_tree = xml.dom.minidom.parse(**'movie.xml'**)  *# 获取所有节点* root = dom\_tree.documentElement  *# 获取sitcom节点* sitcom = root.getElementsByTagName(**'sitcom'**)  *# 循环获取所有sitcom节点的子节点* **for** s **in** sitcom:  *# 获取title节点* title = s.getElementsByTagName(**'title'**)[0]  *# 返回NodeList所以用索引获取*  *# 获取title节点的文本内容* print(**'名称：'**, title.childNodes[0].data)  *# 获取title节点的director属性* print(**'导演：'**, title.getAttribute(**'director'**))   *# 获取players节点* players = s.getElementsByTagName(**'players'**)[0]  *# 循环获取players节点的子节点的文本内容* **for** player **in** players.childNodes:   *# print(player) # 查看节点类型*  *# 只处理元素节点* **if** isinstance(player, xml.dom.minidom.Element):  print(**"演员："**, player.childNodes[0].data)   *# 获取desc节点* **if** s.getElementsByTagName(**'desc'**):  desc = s.getElementsByTagName(**'desc'**)[0]  *# print(desc.childNodes)  # 获取desc节点的文本内容* print(**'详细描述：'**, desc.childNodes[1].data) |
| *movie.xml:*  *<?***xml version="1.0" encoding="UTF-8"***?> <!-- 第一行是XML声明 --> <!-- 这是XML处理指令的例子：处理指令以<?开始,以?>结束 --> <!-- 在<?后的第一个单词是处理指令名,在本例中是xml> --> <!-- 处理指令一定要顶格写,前面不能有任何空白 -->* <**sitcoms**>  <**sitcom**>  <**title director="郑晓龙"**>芈月传</**title**>  <**players**>  <**player**>孙俪</**player**>  <**player**>刘涛</**player**>  <**player**>马苏</**player**>  <**player**>方中信</**player**>  <**player**>黄轩</**player**>  <**player**>高云翔</**player**>  </**players**>  <**desc**>  <![CDATA[战国时期，芈月是楚威王最宠爱的小公主，但<desc>在楚威王死后生活一落千丈，母亲向氏被楚威后逐出宫，  芈月和弟弟芈戎躲过了一次次灾难和危机。芈月与楚公子黄歇青梅竹马，真心相爱，但被作为嫡公主芈姝的陪嫁媵侍远嫁秦国。  芈姝当上了秦国的王后，芈月不得已成为宠妃...]]>  </**desc**>  </**sitcom**>  <**sitcom**>  <**title director="郑晓龙"**>甄嬛传</**title**>  <**players**>  <**player**>孙俪</**player**>  <**player**>陈建斌</**player**>  <**player**>蔡少芬</**player**>  <**player**>蒋欣</**player**>  <**player**>李东学</**player**>  </**players**>  </**sitcom**> </**sitcoms**> |
| *控制台：*  *名称： 芈月传*  *导演： 郑晓龙*  *演员： 孙俪*  *演员： 刘涛*  *演员： 马苏*  *演员： 方中信*  *演员： 黄轩*  *演员： 高云翔*  *详细描述： 战国时期，芈月是楚威王最宠爱的小公主，但<desc>在楚威王死后生活一落千丈，母亲向氏被楚威后逐出宫，*  *芈月和弟弟芈戎躲过了一次次灾难和危机。芈月与楚公子黄歇青梅竹马，真心相爱，但被作为嫡公主芈姝的陪嫁媵侍远嫁秦国。*  *芈姝当上了秦国的王后，芈月不得已成为宠妃...*  *名称： 甄嬛传*  *导演： 郑晓龙*  *演员： 孙俪*  *演员： 陈建斌*  *演员： 蔡少芬*  *演员： 蒋欣*  *演员： 李东学* |

### 3、ElementTree解析

xml.etree.ElementTree模块提供了一个轻量级的API，同时还有一个高效的C语言实现，即xml.etree.cElementTree。与DOM相比，ET的速度更快，API使用更直接、方便。与SAX相比，ET.iterparse函数同样提供了按需解析的功能，不会一次性在内存中读入整个文档。ET的性能与SAX模块大致相仿，但是它的API更加高层次，用户使用起来也很便捷。

#### 3.1、创建xml

创建流程：

1. 通过root = et.Element("xml")创建一个root节点
2. 使用et.SubElement(root, 'FromUserName')创建子节点
3. 给子节点通过text属性填充内容
4. 使用tree = et.ElementTree(root)构建一颗树
5. tree.write('xxx.xml', encoding="utf-8" , xml\_declaration=True)写入文件

##### 方式一

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.etree.cElementTree **as** et  *# 第一步：通过et.Element("xml")创建一个root节点* root = et.Element(**'people'**)  *# 第二步：使用et.SubElement(root, 'FromUserName')创建子节点* person\_element = et.SubElement(root, **'person'**) person\_element2 = et.SubElement(root, **'person'**)  *# 使用et.SubElement(root, 'FromUserName')创建子节点* name\_element = et.SubElement(person\_element, **'name'**) name\_element2 = et.SubElement(person\_element2, **'name'**)  *# 第三步：给子节点通过text属性填充内容* name\_element.text = **'王晶晶'** name\_element2.text = **'至尊宝'** *# 使用et.SubElement(root, 'FromUserName')创建子节点* name\_element = et.SubElement(person\_element, **'age'**) name\_element2 = et.SubElement(person\_element2, **'age'**)  *# 给子节点通过text属性填充内容* name\_element.text = **'28'** name\_element2.text = **'300'** *# 第四步：使用tree = et.ElementTree(root)构建一颗树* tree = et.ElementTree(root)  *# 第五步：调用tree.write('xxx.xml', encoding="utf-8" , xml\_declaration=True)写入文件*  *# xml\_declaration=True 创建xml声明* tree.write(**'people\_create3.xml'**, encoding=**"utf-8"**, xml\_declaration=**True**) |
| *格式化快捷键：Ctrl+Alt+l*  *<?***xml version='1.0' encoding='utf-8'***?>* <**people**>  <**person**>  <**name**>王晶晶</**name**>  <**age**>28</**age**>  </**person**>  <**person**>  <**name**>至尊宝</**name**>  <**age**>28</**age**>  </**person**> </**people**> |

##### 方式二

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.etree.cElementTree **as** et  *# 第一步：通过et.Element("xml")创建一个root节点* root = et.Element(**'people'**)  *# 第二步：使用et.SubElement(root, 'FromUserName')创建子节点* person\_element = et.SubElement(root, **'person'**) person\_element2 = et.SubElement(root, **'person'**)  *# 第三步：给子节点通过attrib设置属性*  *# 一次操作需要设置所有属性，不支持分批添加，后面的操作会覆盖前面的属性* person\_element.attrib = {**'name'**:**'王晶晶'**, **'age'**:**'28'**} person\_element.attrib = dict(no=**'20180620001'**, sex=**'女'**)  *# 如果想要分批操作需要逐个添加，且必须写在字典声明添加和字典函数添加的操作之后，否则数据会丢失 # person\_element.attrib['b'] = 'b' # person\_element.attrib['c'] = 'c' # person\_element.attrib['d'] = 'd' # person\_element.attrib = {'name':'王晶晶', 'age':'28'}* person\_element2.attrib[**'name'**] = **'至尊宝'** person\_element2.attrib[**'age'**] = **'300'** *# 第四步：使用tree = et.ElementTree(root)构建一颗树* tree = et.ElementTree(root)  *# 第五步：调用tree.write('xxx.xml', encoding="utf-8" , xml\_declaration=True)写入文件 # xml\_declaration=True 创建xml声明 # short\_empty\_elements=False True闭合标签，False成对标签* tree.write(**'people\_create4.xml'**, encoding=**"utf-8"**, xml\_declaration=**True**, short\_empty\_elements=**False**) |
| *<?***xml version='1.0' encoding='utf-8'***?>* <**people**>  <**person no="20180620001" sex="女"**></**person**>  <**person age="300" name="至尊宝"**></**person**> </**people**> |

#### 3.2、解析xml

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** xml.etree.cElementTree **as** et  *# 将xml文档解析为tree树* tree = et.parse(**"people\_create3.xml"**)  *# 获取根节点* root = tree.getroot()  *# 遍历根节点的所有子节点* **for** p **in** root:  *# print(p.tag)  # 添加sex节点* sex\_element = et.Element(**'sex'**)  sex\_element.text = **'男'** p.append(sex\_element)  **for** tName **in** p:  *# 修改name节点* **if 'name'** == tName.tag:  *# print(tName.tag, tName.text)* **if '王晶晶'** == tName.text:  tName.text = **'张三'** tName.attrib[**'id'**] = **'111'** tName.attrib[**'no'**] = **'20180523001'** *# tName.attrib = dict(no='20180523001') # 添加唯一属性* **if '至尊宝'** == tName.text:  tName.text = **'李四'** tName.attrib[**'id'**] = **'222'** tName.attrib[**'no'**] = **'20180523002'** *# tName.attrib = dict(no='20180523002') # 添加唯一属性   # 删除age节点* **if 'age'** == tName.tag:  *# print(tName.tag, tName.text)* p.remove(tName)   *# 更新文件* tree.write(**'people\_create3.xml'**, encoding=**"utf-8"**, xml\_declaration=**True**) |
| *<?***xml version='1.0' encoding='utf-8'***?>* <**people**>  <**person**>  <**name id="111" no="20180523001"**>张三</**name**>  <**sex**>男</**sex**>  </**person**>  <**person**>  <**name id="222" no="20180523002"**>李四</**name**>  <**sex**>男</**sex**>  </**person**> </**people**> |

## XPath

先安装lxml

管理员方式打开cmd，然后输入pip install lxml安装即可

### XPath爬取网页信息

xpath\_xml.py

|  |
| --- |
| *# -\*- coding:utf-8 -\*-* **import** urllib.request **as** request **from** lxml **import** etree  *# 模拟浏览器访问地址，并且接受返回的数据* resp = request.urlopen(**'http://www.mp4ba.net/'**)  *# 将字节数据转化成字符串(整个页面的内容)* html\_content = str(resp.read(), encoding=**'gbk'**)  *# 根据字符串获取xml树* tree = etree.HTML(html\_content)  *# 获取页面上id为normalthread\_开头的节点信息 # results = tree.xpath('//\*[@id="normalthread\_5989"]/tr/th/a[1]')* results = tree.xpath(**'//\*[starts-with(@id, "normalthread\_")]/tr/th'**)   **for** item **in** results:  **try**:  first = item.xpath(**'./em/a/text()'**)[0]  print(first)  second = item.xpath(**'./a[@class="s xst"]/text()'**)[0]  print(second)  \_i = item.xpath(**'./div[@class="foruminfo"]/i[@class="z"]'**)[0]  \_text = \_i.xpath(**'./a/span/text()'**)[0]  print(\_text)  **except** Exception **as** e:  print(**'解析页面元素错误，错误信息为：'**, e) |

