

# Python3 XML 解析

## 什么是 XML ?

XML 指可扩展标记语言（eXtensible Markup Language），标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。你可以通过本站学习 [XML 教程](#)

XML 被设计用来传输和存储数据。

XML 是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。

它也是元标记语言，即定义了用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。

## 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。

本章节使用到的 XML 实例文件 **movies.xml** 内容如下：

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
  <type>War, Thriller</type>
  <format>DVD</format>
  <year>2003</year>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
  <type>Anime, Science Fiction</type>
  <format>DVD</format>
  <year>1989</year>
  <rating>R</rating>
  <stars>8</stars>
  <description>A schientific fiction</description>
</movie>
  <movie title="Trigun">
    <type>Anime, Action</type>
```

```
<format>DVD</format>
<episodes>4</episodes>
<rating>PG</rating>
<stars>10</stars>
<description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
  <type>Comedy</type>
  <format>VHS</format>
  <rating>PG</rating>
  <stars>2</stars>
  <description>Viewable boredom</description>
</movie>
</collection>
```

## Python 使用 SAX 解析 xml

SAX 是一种基于事件驱动的API。

利用 SAX 解析 XML 文档牵涉到两个部分: **解析器**和**事件处理器**。

解析器负责读取 XML 文档, 并向事件处理器发送事件, 如元素开始跟元素结束事件。

而事件处理器则负责对事件作出响应, 对传递的 XML 数据进行处理。

- 1、对大型文件进行处理;
- 2、只需要文件的部分内容, 或者只需从文件中得到特定信息。
- 3、想建立自己的对象模型的时候。

在 Python 中使用 sax 方式处理 xml 要先引入 xml.sax 中的 parse 函数, 还有 xml.sax.handler 中的 ContentHandler。

### ContentHandler 类方法介绍

#### characters(content) 方法

调用时机:

从行开始, 遇到标签之前, 存在字符, content 的值为这些字符串。

从一个标签, 遇到下一个标签之前, 存在字符, content 的值为这些字符串。

从一个标签, 遇到行结束符之前, 存在字符, content 的值为这些字符串。

标签可以是开始标签, 也可以是结束标签。

#### startDocument() 方法

文档启动的时候调用。

#### endDocument() 方法

解析器到达文档结尾时调用。

#### startElement(name, attrs) 方法

遇到XML开始标签时调用, name 是标签的名字, attrs 是标签的属性值字典。

#### endElement(name) 方法

遇到XML结束标签时调用。

## make\_parser 方法

以下方法创建一个新的解析器对象并返回。

```
xml.sax.make_parser( [parser_list] )
```

参数说明:

- **parser\_list** - 可选参数，解析器列表

## parser 方法

以下方法创建一个 SAX 解析器并解析xml文档：

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler])
```

参数说明:

- **xmlfile** - xml文件名
- **contenthandler** - 必须是一个 ContentHandler 的对象
- **errorhandler** - 如果指定该参数，errorhandler 必须是一个 SAX ErrorHandler 对象

## parseString 方法

parseString 方法创建一个 XML 解析器并解析 xml 字符串：

```
xml.sax.parseString(xmlstring, contenthandler[, errorhandler])
```

参数说明:

- **xmlstring** - xml字符串
- **contenthandler** - 必须是一个 ContentHandler 的对象
- **errorhandler** - 如果指定该参数，errorhandler 必须是一个 SAX ErrorHandler对象

## Python 解析XML实例

```
#!/usr/bin/python3

import xml.sax

class MovieHandler( xml.sax.ContentHandler ):
    def __init__( self ):
        self.reset()
    def reset( self ):
        self.chapters = []
    def startElement( self, namespace, name, attrs, instance ):
        if name == 'chapter':
            self.chapters.append( instance )
    def endElement( self, namespace, name, instance ):
        if name == 'chapter':
            instance = None
    def characters( self, content ):
        pass
```

```
def __init__(self):
    self.CurrentData = ""
    self.type = ""
    self.format = ""
    self.year = ""
    self.rating = ""
    self.stars = ""
    self.description = ""

# 元素开始调用
def startElement(self, tag, attributes):
    self.CurrentData = tag
    if tag == "movie":
        print ("*****Movie*****")
        title = attributes["title"]
        print ("Title:", title)

# 元素结束调用
def endElement(self, tag):
    if self.CurrentData == "type":
        print ("Type:", self.type)
    elif self.CurrentData == "format":
        print ("Format:", self.format)
    elif self.CurrentData == "year":
        print ("Year:", self.year)
    elif self.CurrentData == "rating":
        print ("Rating:", self.rating)
    elif self.CurrentData == "stars":
        print ("Stars:", self.stars)
    elif self.CurrentData == "description":
        print ("Description:", self.description)
    self.CurrentData = ""

# 读取字符时调用
def characters(self, content):
    if self.CurrentData == "type":
        self.type = content
    elif self.CurrentData == "format":
        self.format = content
    elif self.CurrentData == "year":
        self.year = content
    elif self.CurrentData == "rating":
        self.rating = content
    elif self.CurrentData == "stars":
        self.stars = content
    elif self.CurrentData == "description":
        self.description = content
```

```
if ( __name__ == "__main__"):
```

```
# 创建一个 XMLReader
parser = xml.sax.make_parser()
# 关闭命名空间
parser.setFeature(xml.sax.handler.feature_namespaces, 0)

# 重写 ContextHandler
Handler = MovieHandler()
parser.setContentHandler( Handler )

parser.parse("movies.xml")
```

以上代码执行结果如下：

```
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Year: 1989
Rating: R
Stars: 8
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Stars: 10
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Stars: 2
Description: Viewable boredom
```

完整的 SAX API 文档请查阅[Python SAX APIs](#)

## 使用xml.dom解析xml

文件对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展置标语言的标准编程接口。

一个 DOM 的解析器在解析一个 XML 文档时，一次性读取整个文档，把文档中所有元素保存在内存中的一个树结构里，之后你可以利用DOM 提供的不同的函数来读取或修改文档的内容和结构，也可以把修改过的内容写入xml文件。

python中用xml.dom.minidom来解析xml文件，实例如下：

```
#!/usr/bin/python3

from xml.dom.minidom import parse
import xml.dom.minidom

# 使用minidom解析器打开 XML 文档
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
    print ("Root element : %s" % collection.getAttribute("shelf"))

# 在集合中获取所有电影
movies = collection.getElementsByTagName("movie")

# 打印每部电影的详细信息
for movie in movies:
    print ("*****Movie*****")
    if movie.hasAttribute("title"):
        print ("Title: %s" % movie.getAttribute("title"))

    type = movie.getElementsByTagName('type')[0]
    print ("Type: %s" % type.childNodes[0].data)
    format = movie.getElementsByTagName('format')[0]
    print ("Format: %s" % format.childNodes[0].data)
    rating = movie.getElementsByTagName('rating')[0]
    print ("Rating: %s" % rating.childNodes[0].data)
    description = movie.getElementsByTagName('description')[0]
    print ("Description: %s" % description.childNodes[0].data)
```

以上程序执行结果如下：

```
Root element : New Arrivals
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Rating: PG
Description: Talk about a US-Japan war
*****Movie*****
```

```
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Description: Viewable boredom
```

完整的 DOM API 文档请查阅[Python DOM APIs](#)。

[← Python3 多线程](#)

[Python3 JSON 数据解析 →](#)

[✎ 点我分享笔记](#)