



3.6 Python解析XML

3.6.1 XML语言概述

3.6.2 XML文档解析

3.6.3 Python解析XML



3.6.1 XML语言概述

- XML:可扩展标记语言 (Extensible Markup Language)
- XML是W3C的推荐标准
- XML用途：应用于 Web 开发的许多方面，常用于简化数据的存储和共享
 - XML 把数据从 HTML 分离
 - XML 简化数据共享
 - XML 数据以纯文本格式进行存储，提供了一种独立于软件和硬件的数据存储方法。
 - XML 简化平台变更
 - XML 使您的数据更有用，通过 XML，您的数据可供各种阅读设备使用（掌上计算机、语音设备、新闻阅读器等），还可以供盲人或其他残障人士使用。
 - XML 简化数据传输



3.6.1 XML语言概述

- XML用途（续）：

- XML 用于创建新的互联网语言

- XHTML

- 用于描述可用的 Web 服务 的 WSDL
- 作为手持设备的标记语言的 WAP 和 WML
- 用于新闻 feed 的 RSS 语言
- 描述资本和本体的 RDF 和 OWL
- 用于描述针对 Web 的多媒体 的 SMIL



3.6.1 XML语言概述

```
<collection shelf="New Arrivals">
  <works title="电影">
    <names>敦刻尔克</names>
    <author>诺兰</author>
  </works>
  <works title="书籍">
    <names>我的职业是小说家</names>
    <author>村上春树</author>
  </works>
</collection>
```

■ XML特点:

1. 标签可扩展性（自定义标签）
2. 数据和显示相分离
3. 自描述性
4. 具有良好的格式
5. 保值性



3.6.1 XML语言概述

- 定义、标识、验证、显示、解析XML文档技术：
 - 定义（验证）XML文档有效：
 - DTD（Document Type Definition，文档类型定义）
 - XML Schema模式(XML Schema Definition (XSD))
 - 标识XML文档标签唯一：
 - XML命名空间
 - 显示XML文档
 - CSS层叠样式表（Cascading Style Sheet）
 - XSL可扩展样式语言 (eXtensible Stylesheet Language)
 - 解析XML文档
 - DOM、SAX、JDOM、DOM4J



3.6 XML语言

3.6.1 XML语言概述

3.6.2 XML文档解析

3.6.3 Python解析XML

3.6.2 XML文档解析

■ XML解析

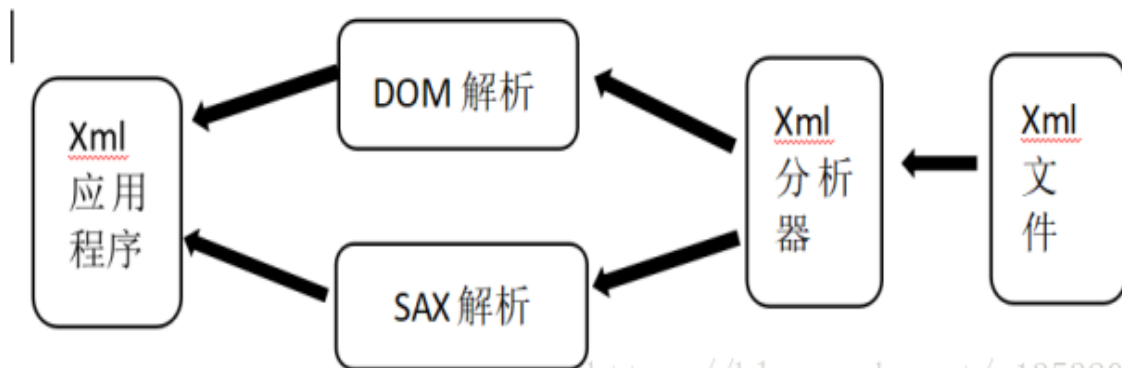
- 按照XML中的元素取出对应的信息

■ XML解析器：

- 从XML文件中获得信息和数据的程序

■ XML解析方式：

- DOM解析
- SAX解析
- DOM4J解析
- JDOM解析
-



<https://blog.csdn.net/a1353206432>

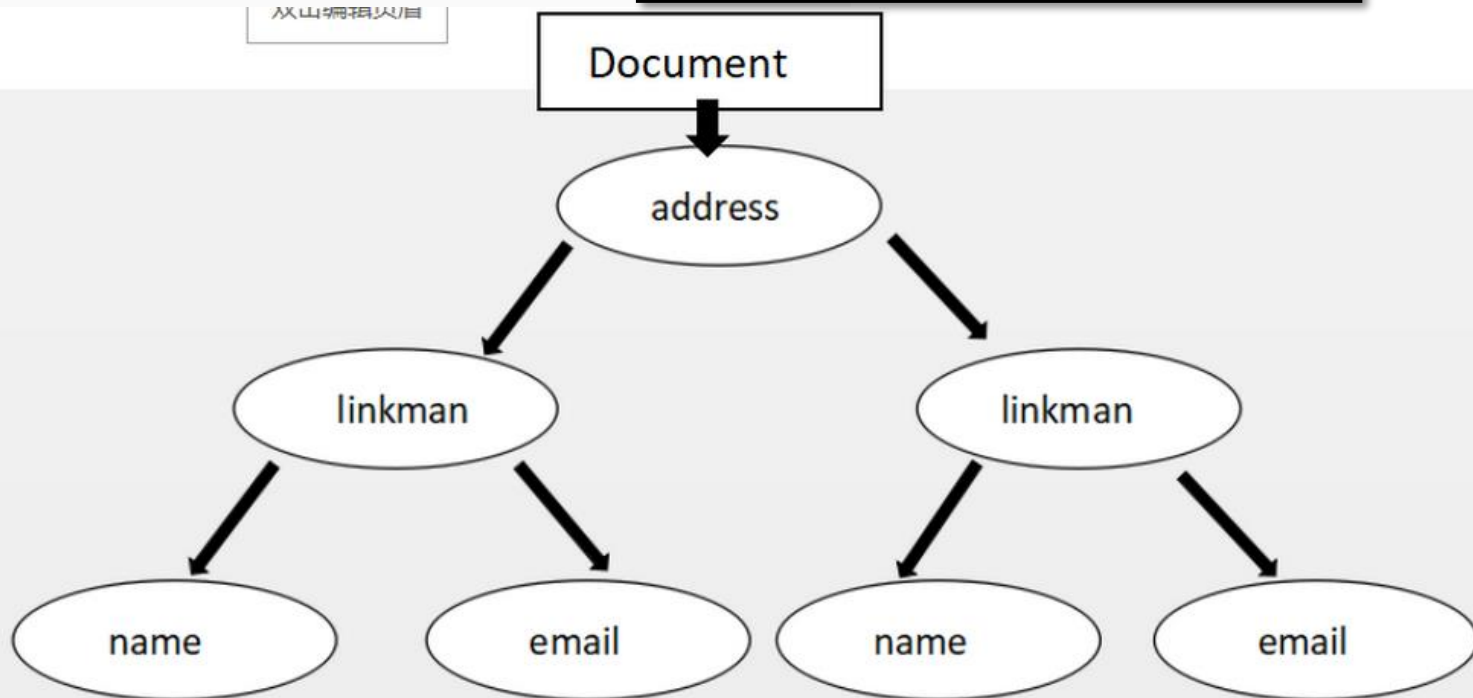


3.6.2 XML文档解析

- DOM解析（Document Object Model，文档对象模型）
 - https://blog.csdn.net/cui_shen/article/details/90813385
 - <https://blog.csdn.net/a1353206432/article/details/80583302>
- W3C 的规范（<http://www.w3.org/DOM/>），是一种与浏览器、平台、语言无关的接口，可用于解析HTML、XML
- 解析器将一个 XML文档转换成一个对象模型的集合（DOM 树）
- 应用程序通过对这个对象模型的操作，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，也将按树的形式进行转换。


```
<?xml version="1.0" encoding="GBK"?>
<address>
  <linkman>
    <name>Van_DarkHolme</name>
    <email>van_darkholme@163.com</email>
  </linkman>
  <linkman>
    <name>Bili</name>
    <email>Bili@163.com</email>
  </linkma
</address>
```

将该xml转换为树的结构为:





3.6.2 XML文档解析

- DOM通过一系列接口来表达文档对象，四个基本的接口：
 - Document:
 - 代表整个xml文档，表示为整个DOM的根，即为该树的入口，通过该接口可以访问xml中所有元素的内容
 - Node:
 - 代表一个DOM树节点，是其他大多数接口的父类
 - NodeList:
 - 表示一个节点的集合，包含了某个节点中的所有子节点。
 - NamedNodeMap:
 - 一个节点的集合，通过该接口，可以建立节点名和节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点

3.6.2 XML文档解析

- 使用DOM解析器解析XML文档至少需要三步：

- 1. 建立一个解析器工厂

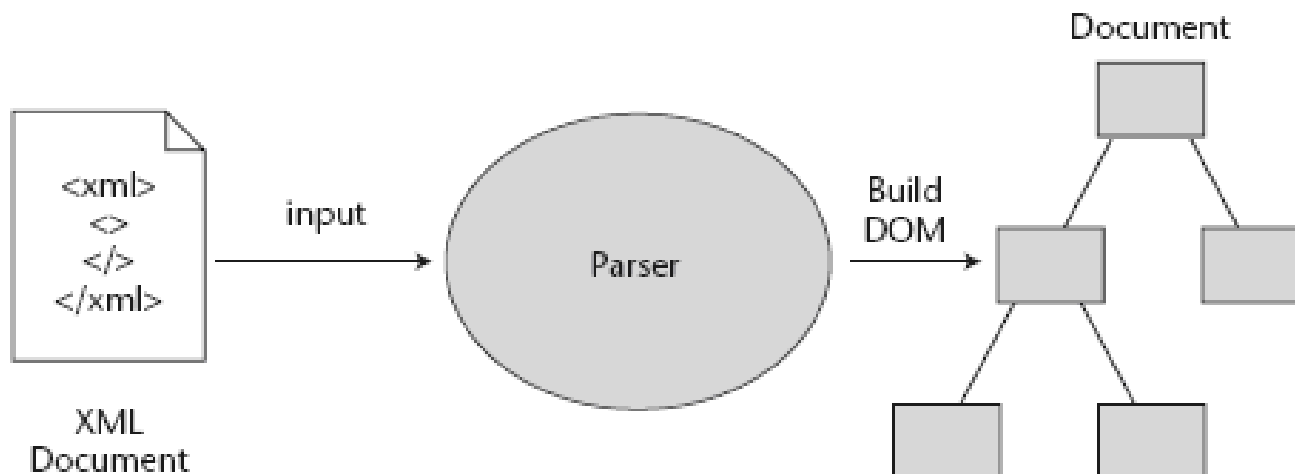
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

- 2. 创建一个解析器对象

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

- 3. 将解析器和XML文档联系起来，开始解析

```
Document doc = builder.parse(new File(filename));
```



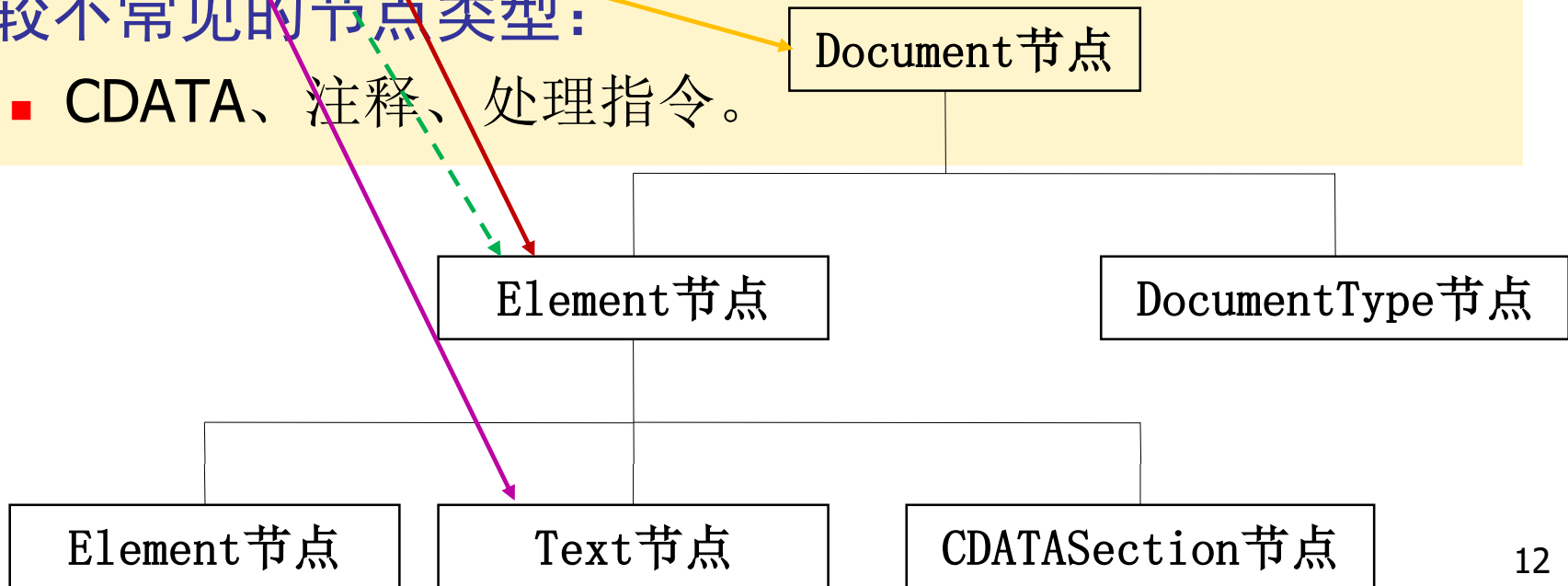
■ DOM模型结构最常见的对象节点类型

`<div id="box">啊哈哈哈哈哈</div>`

- 元素：元素是XML的基本构件。
 - 元素可以有其他元素、文本节点或两者兼有来作为其子节点
 - 元素节点还是可以有属性的唯一类型的节点
- 属性：属性节点包含关于元素节点的信息
- 文本：文本节点是文本。
- 文档（根节点）：文档节点是整个文档中所有其他节点的父节点。

■ 较不常见的节点类型：

- CDATA、注释、处理指令。



```
<bookstore>
```

```
<book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

```
<book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
```

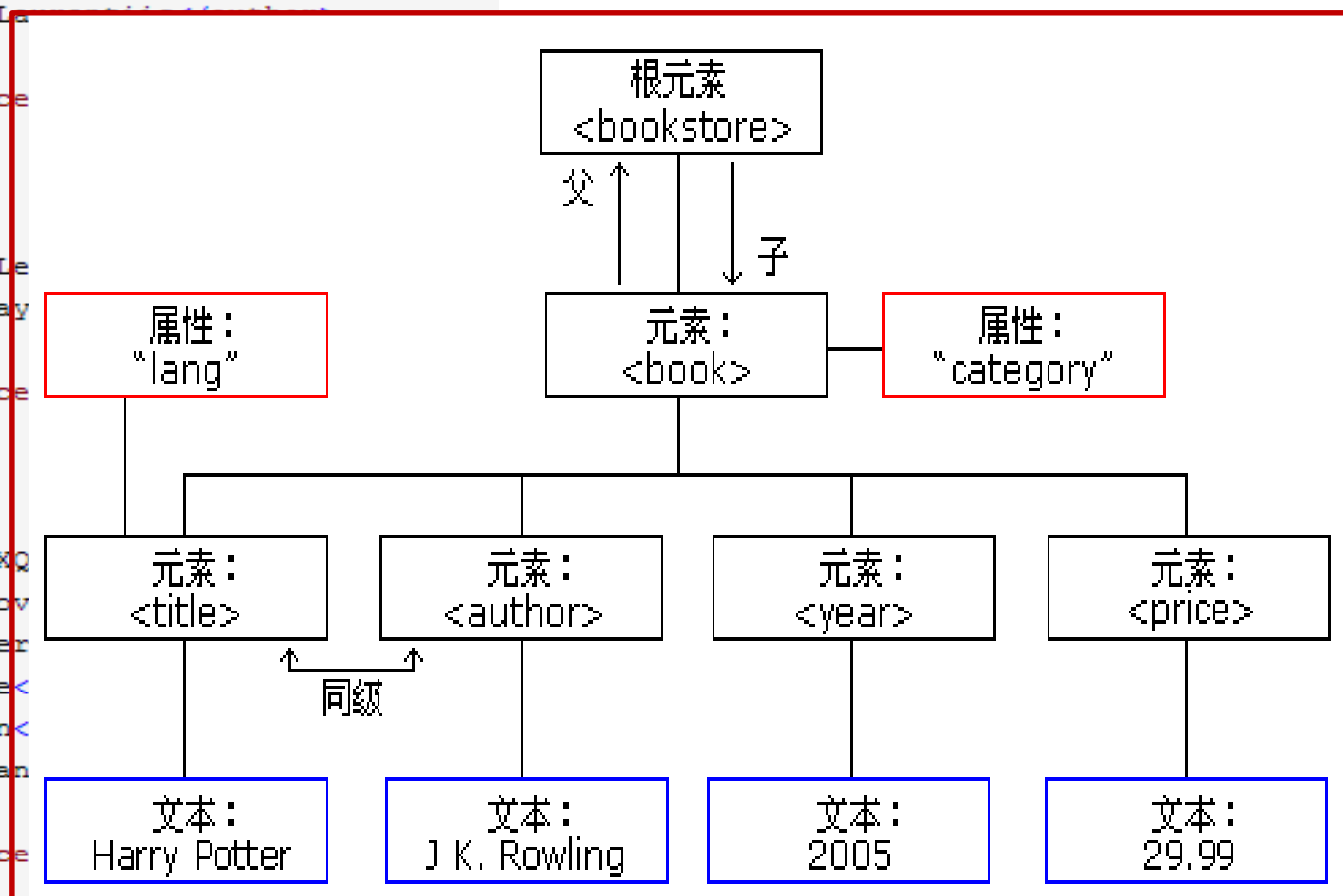
```
<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
```

```
<book category="web">
  <title lang="en">XQuery</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Gopalakrishnan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
```

```
</bookstore>
```

XML文档

解析为
DOM树





3.6.2 XML文档解析

■ Document对象节点常用的方法

方法	功能描述
<code>getDocumentElement()</code>	返回当前节点的Element节点
<code>getDoctype()</code>	返回当前节点的DocumentType子节点
<code>getElementByTagName(String name)</code>	返回一个NodeList对象，该对象由参数name指定的节点的Element类型子孙节点组成。
<code>getElementByTagNameNS(String namespaceURI, String localname)</code>	返回一个NodeList对象，该对象由参数localname指定的节点的Element类型子孙节点组成，localname的名称空间由参数namespaceURI指定

Element对象方法

方法

功能描述

getTagName()

返回该节点的名称，即对应XML文件的标记名称

getAttribute(String name)

返回该节点的对应属性的值，属性名称由参数name指定。即对应XML文件标记的属性值

getElementByTagName(String name)

返回一个NodeList对象，该对象由该节点的Element类型的子孙节点组成，子孙节点的名称由参数name指定

getElementByTagNameNs(String namespaceURI, String localName)

返回一个NodeList对象，该对象由该节点的Element类型的子孙节点组成，子孙节点的名称由参数localName指定，名称空间由参数namespaceURI指定

getAttributeNode(String name)

返回属性节点

hasAttribute(String name)

判断该节点是否具有某个属性，属性名称由参数name指定

hasAttributeNS(String namespaceURI, String localName)

判断该节点是否具有某个属性，属性名称由参数name指定，名称空间由参数namespaceURI指定

removeAttribute(String name)

删除name属性

removeAttributeNode(Attr oldAttr)

删除当前节点的oldAttr属性

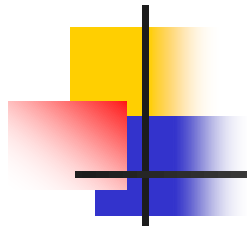
setAttribute(String name,String value)

设置name属性的值

setAttributeNode(Attr newAttr)

为当前的节点设置属性

解析XML文档示例



```
import javax.xml.parsers.*;
import org.w3c.dom.*;
public class DomDemo1
{
```

```
    public static void main(String args[]) throws Exception
    {
```

```
        Document doc=null;
        DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
        DocumentBuilder db=dbf.newDocumentBuilder();
        doc=db.parse("students.xml");
        Element root=doc.getDocumentElement(); //获取文档的根元素
        //得到第一个sname结点
        Node sname=root.getElementsByTagName("sname").item(0);
        //输出第一个学生的姓名
        System.out.println(sname.getChildNodes().item(0).getNodeValue());
    }
}
```

```
<students>
```

```
  <student sex="男" email="zhanger@126.com"
    cellphone="13812345678">
```

```
    <sno>1001</sno>
```

```
    <sname>张小二</sname>
```

```
    <sclass>软件091</sclass>
```

```
    <birthday>1991-07-03</birthday>
```

```
  </student>
```

```
  <!--其它student元素-->
```

```
</students>
```


添加元素示例代码

//引入相应的包！！

```
import javax.xml
```

```
import org.w3c
```

```
import javax.xml
```

```
import javax.xml.transform.dom.*;
```

```
import javax.xml.transform.stream.*;
```

```
import java.io.*;
```

新生成的**student**子元素

<student><sname>李小军</sname></student>

```
Element student=doc.createElement("student");
```

```
Element sname=doc.createElement("sname");
```

```
sname.setTextContent("李小军");
```

```
student.appendChild(sname);
```

```
root.appendChild(student); //root为根元素Students
```

```
TransformerFactory tff=TransformerFactory.newInstance();
```

```
Transformer tf=tff.newTransformer();
```

```
tf.transform(new DOMSource(doc),
```

```
    new StreamResult(new File("stude1.xml"))));
```



3.6.2 XML文档解析

■ DOM编程

- 目前有多种语言的XML解析器和库，包括
JAVA、C++、Python
- 多种形式：
 - 以部件的形式提供，如MSXML
 - JAVA实现，IBM公司XML4J
 - JAVA实现，SUN公司Project X



3.6 XML语言

3.6.1 XML语言概述

3.6.2 XML文档解析

3.6.3 Python解析XML

3.6.3 Python解析XML

- Python的标准库中，提供了多种可以用于处理XML的包：
 - `xml.dom`
 - `xml.dom`实现的是W3C制定的DOM API
 - `xml.dom.minidom`
 - `xml.dom.minidom`是DOM API的极简化实现
 - `xml.dom.pulldom`
 - 提供一个“pull解析器”
 - `xml.sax`
 - `xml.sax`模块实现的是SAX API，并不是由W3C官方所提出的标准。它是事件驱动的，并不需要一次性读入整个文档
 - `xml.parser.expat`
 - `xml.parser.expat`提供了对C语言编写的expat解析器的一个直接的、底层API接口。expat接口与SAX类似，也是基于事件回调机制，但是这个接口并不是标准化的，只适用于expat库。
 - `xml.etree.ElementTree`（以下简称ET）

XML文档mov.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <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xml.dom解析
xml<https://www.cnblogs.com/kumata/p/9447732.html>

目录层次:

collect shelf

movie title

XXXX

XXXX

■ Python代码

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

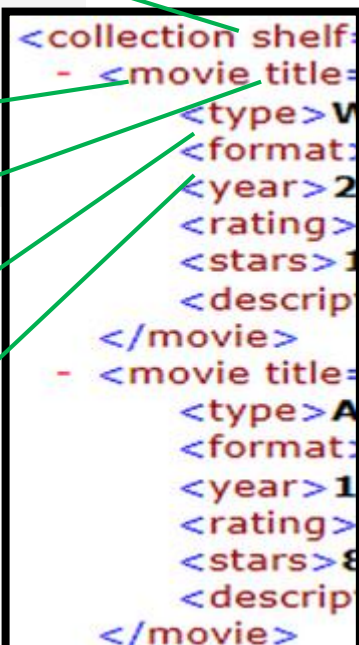
#使用Minidom解析器打开xml文档
DOMTree = xml.dom.minidom.parse("mov.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
    print("Root element : %s " % collection.getAttribute("shelf"))

#self.getElementsByTagName("xx")取下级
#在集合里面获取所有电影
movies = collection.getElementsByTagName("movie")

#打印每部电影的详细信息
for movie in movies:
    print("*****movies*****")
    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)
```



```
<collection shelf="V">
  - <movie title="The Matrix" type="V" format="DVD" year="2003" rating="4.5" stars="10" description="The Matrix" />
  - <movie title="The Matrix Reloaded" type="A" format="DVD" year="2003" rating="4.5" stars="8" description="The Matrix Reloaded" />
</collection>
```

■ Python代码（续）

#有些电影里不一定有year这条信息，用if语句不会导致报错

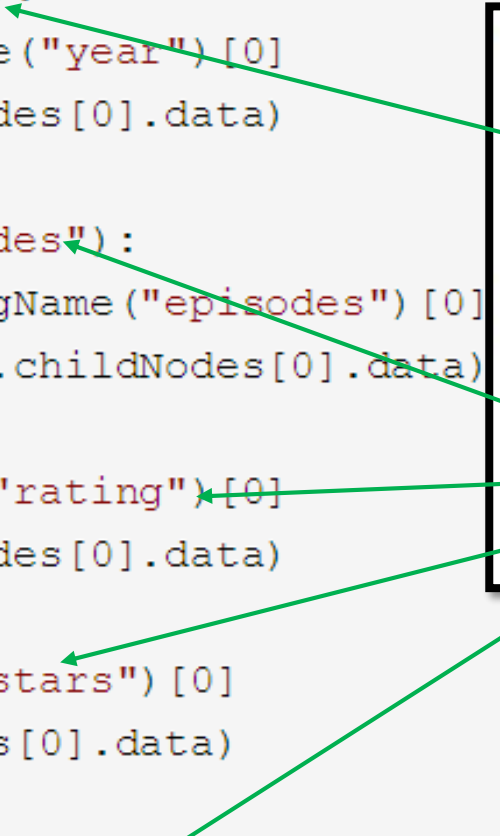
```
if movie.getElementsByTagName("year"):
    year = movie.getElementsByTagName("year")[0]
    print("year : %s" % year.childNodes[0].data)

if movie.getElementsByTagName("episodes"):
    episodes = movie.getElementsByTagName("episodes")[0]
    print("episodes : %s" % episodes.childNodes[0].data)

rating = movie.getElementsByTagName("rating")[0]
print("rating : %s" % rating.childNodes[0].data)

stars = movie.getElementsByTagName("stars")[0]
print("stars : %s" % stars.childNodes[0].data)

description = movie.getElementsByTagName("description")[0]
print("description : %s" % description.childNodes[0].data)
```



The diagram illustrates the mapping between XML data and Python code. It features two XML snippets. The first snippet contains tags for title, type, format, year, rating, stars, and description. The second snippet contains tags for title, type, format, episodes, rating, stars, and description. Green arrows originate from specific tags in these snippets and point to corresponding variables in the Python code on the left. Specifically, arrows point from the 'year' tag to the 'year' variable, from the 'episodes' tag to the 'episodes' variable, from the 'rating' tag to the 'rating' variable, from the 'stars' tag to the 'stars' variable, and from the 'description' tag to the 'description' variable.

```
<movie title="T
<type>Ani
<format>DV
<year>198
<rating>R<
<stars>8</
<description
</movie>
<movie title="T
<type>Ani
<format>DV
<episodes>
<rating>PG
<stars>10<
<description
```



```
Root element : New Arrivals
*****movies*****
title : Enemy Behind
type : War, Thriller
Format : DVD
year : 2003
rating : PG
stars : 10
description : Talk about a US-Japan war
*****movies*****
title : Transformers
type : Anime, Science Fiction
Format : DVD
year : 1989
rating : R
stars : 8
description : A schientific fiction
```

■ 输出结果

```
*****movies*****
title : Trigun
type : Anime, Action
Format : DVD
episodes : 4
rating : PG
stars : 10
description : Vash the Stampede!
*****movies*****
title : Ishtar
type : Comedy
Format : VHS
rating : PG
stars : 2
description : Viewable boredom
```


3.6.3 Python解析XML

- 例.Python解析XML文件
- <https://zhuanlan.zhihu.com/p/29219620>
- 本章节使用的XML实例文件works.xml内容如下：

```
<collection shelf="New Arrivals">
<works title="电影">
  <names>敦刻尔克</names>
  <author>诺兰</author>
</works>
<works title="书籍">
  <names>我的职业是小说家</names>
  <author>村上春树</author>
</works>
</collection>
```

解析结果

内容

类型： 电影

名称： 敦刻尔克

作者： 诺兰

内容

类型： 书籍

名称： 我的职业是小说家

作者： 村上春树

Python使用SAX接口解析XML

```
#!/usr/bin/python3
#encoding=utf-8

import xml.sax
import xml.sax.handler
```

```
if (__name__ == "__main__"):
    #创建一个XMLReader
    parser = xml.sax.make_parser()
    parser.setFeature(xml.sax.handler.feature_namespaces,0)
    #重写ContextHandler
    Handler = WorksHandler()
    parser.setContentHandler(Handler)
    parser.parse("works.xml")
```

```
class WorksHandler(xml.sax.ContentHandler):
    def __init__(self):
        self.CurrentData = ""
        self.names = ""
        self.author = ""
    #元素开始事件处理
    def startElement(self, tag, attributes):
        self.CurrentData = tag
        if tag == "works":
            print("***内容***")
            title = attributes["title"]
            print("类型: ",title)
```

```
    #元素结束事件处理
    def endElement(self, tag):
        if self.CurrentData == "names":
            print("名称: ",self.names)
        elif self.CurrentData == "author":
            print("作者: ",self.author)
        self.CurrentData = ""
    #内容事件处理
    def characters(self, content):
        if self.CurrentData == "names":
            self.names = content
        elif self.CurrentData == "author":
            self.author = content
```