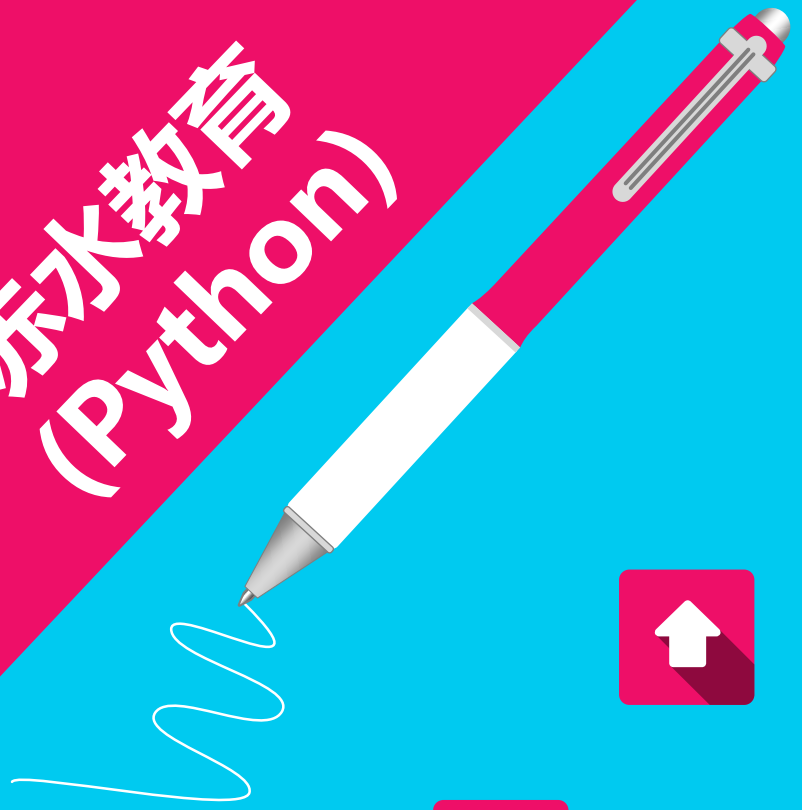
赤水教育

# Python零基础入门到开发

视频教程入门到精通

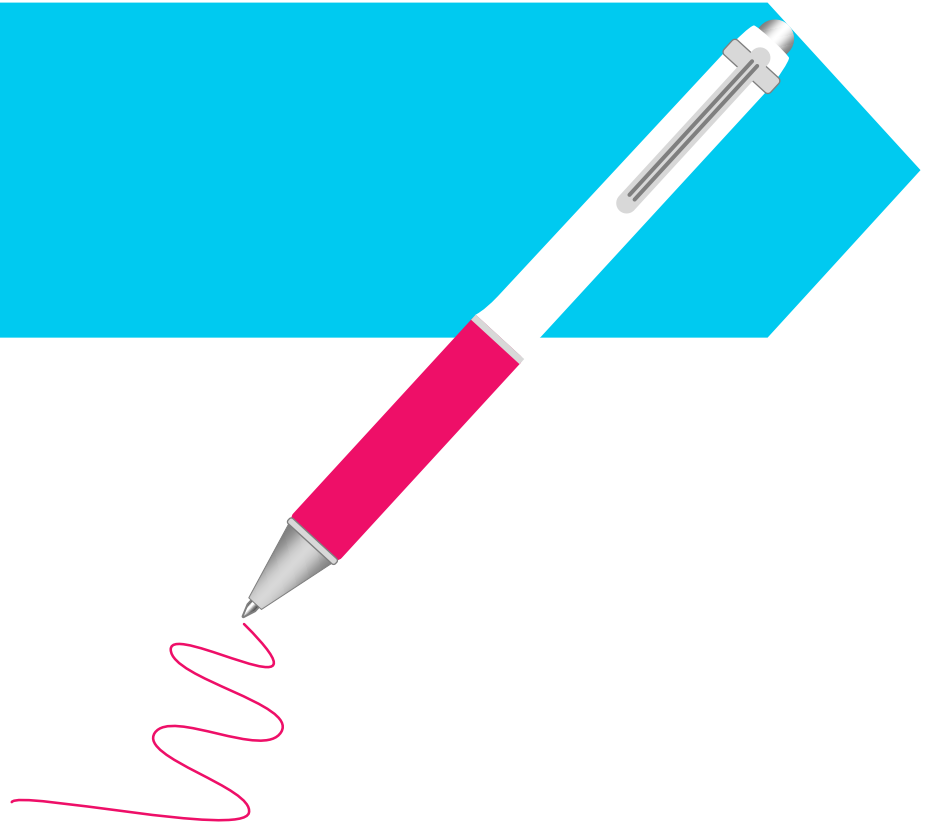
赤水教育教育出品

# 赤水教育（Python）



- XML介绍
- Sax解析XML
- Dom解析XML
- Etree解析XML

# 1 XML介绍



# 1.1 XML介绍

XML:
指可扩展标记语言，被设计用来传输和存储数据，已经日趋成为当前许多新生技术的核心，在不同的领域都有着不同的应用。它是web发展到一定阶段的必然产物，既具有SGML的核心特征，又有着HTML的简单特性，还具有明确和结构良好等许多新的特性。

xml指可扩展标记语言（EXtensible Markup Language）
xml是一种标记语言，类似html
xml的设计宗旨是存储和传输数据，和html不同，html是用来主攻显示数据
xml标签没有被预定义。您需要自定义标签
xml被设计为具有自我描述性
xml是w3c的推荐标准

xml和html的对比:

XML 不是 HTML 的替代。
XML 和 HTML 为不同的目的而设计：
XML 被设计为传输和存储数据，其焦点是数据的内容。
HTML 被设计用来显示数据，其焦点是数据的外观。
HTML 旨在显示信息，而 XML 旨在传输信息。

## XML示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<people><!--根元素(根节点,也叫root节点)-->
        <name>Bob</name><!--子元素(子节点)-->
        <sex>Male</sex>
        <old>16</old>
</people>
```

**XML 文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端。**
**父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。**
**XML 文档必须包含根元素。该元素是所有其他元素的父元素。**

**1、所有元素均可拥有子元素**
**2、所有元素均可拥有文本内容和属性（类似 HTML ）**
**3、所有的XML元素都必须有关闭标签**
**4、xml标签对大小写敏感**
**5、xml标签必须正确的嵌套**
**6、xml的属性必须加引号**
**7、在 XML 中，空格会被保留，多个空格被解释称一个**

**针对元数据的 XML 属性**

**有时候会向元素分配 ID 引用。这些 ID 索引可用于标识 XML 元素，它起作用的方式与 HTML 中 ID 属性是一样的。这个例子向我们演示了这种情况：**

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
        <people id ='1' >
                <name>Bob</name>
                <sex>Male</sex>
                <old>16</old>
        </people>
        <people id ='2' >
                <name>Lina</name>
                <sex>Female</sex>
                <old>30</old>
        </people>
</person>
```

**上面的 ID 仅仅是一个标识符，用于标识不同的便签。它并不是便签数据的组成部分。**

**实体引用**
**在 XML 中，一些字符拥有特殊的意义。**
**如果你把字符 "<" 放在 XML 元素中，会发生错误，这是因为解析器会把它当作新元素的开始。**
**这样会产生 XML 错误：**

<message> if salary < 1000 </message>

| | | |
|---|---|---|
| < | < | 小于 |
| > | > | 大于 |
| & | & | 和号 |
| ' | ' | 单引号 |
| " | " | 引号 |

**修改之后：**

<message> if salary < 1000 </message>

# 2 SAX解析XML

# 2.1 Sax解析XML

**SAX是一种基于事件解析XML的方式：(事件触发，处理函数被调用)**

**python 标准库包含SAX解析器，SAX是一种典型的极为快速的工具，在解析XML时，不会占用大量内存。但是这是基于回调机制的，因此在某些数据中，它会调用某些方法进行传递。这意味着必须为数据指定句柄，以维持自己的状态，这是非常困难的。**

**在xml中，事件与用户操作无关，而与正在读取的xml文档中的元素有关，每当它读到xml中的内容时，他都会生成一系列的事件。**

**Sax是流接口，因为文档是通过连续的字节流一次处理一行，因此你不能访问xml文档中的你想访问的部分，也不能回溯。这样的方式在内存操作方面更有效率。**

# 2.2 Sax API介绍

import xml.sax

导入模块

startDocument()方法：

文档启用的时候调用

endDocument()方法：

解释器到达文档结尾时调用

startElement(name.attrs)方法：

遇到开始标签时候调用

endElement(name)方法：

遇到结束标签时触发

characters(content)：

遇到字符串时触发

#遇到xml结束标签时调用解析器负责读取xml文档，并产生事件，（比如标签开始和标签结束），而事件处理器负责对事件，做出相应处理。

# 2.3 解析代码示例

**1、创建一个xml解析器：**

```
parser = xml.sax.make_parser()
```

**2、初始化我们继承重写的类对象：**

```
class PeopleHandler(xml.sax.ContentHandler):
        def startElement(self, tag, attributes)
            #遇到开始标签时触发，attributes 标签属性字典
        def characters(self,content)
            #内容事件处理，遇到字符串时触发
        def endElement(self,tag)
            #遇到结束标签时触发
Handler = PeopleHandler()
parser.setContentHandler(Handler)
```

**3、解析XML文档：**

```
parser.parse('test.xml')
```

**具体代码：**

```
# -*- coding:utf-8 -*-
class PeopleHandler(xml.sax.ContentHandler):
        def __init__(self):
        #初始化构造函数
                self.CurrentData=""
                #用来保存目前你要处理的xml标签,通过这个数据对象
                #我们可以在获取到不同的标签时，做不同的处理
                self.age=""
                self.sex=""
                self.salary=""
```

**注：这些函数都包含在我们继承重写的sax类对象中！**

```
def startElement(self, tag, attributes):
#处理开始标签,attributes 标签属性字典
#只有开始标签有属性
        self.CurrentData = tag
        if tag == "people":
                #标签是people的时候
                #那么代表我们的标签开始处理
                print('-------------')
                name = attributes["name"]
                #通过标签属性字典 来获取标签的属性
                print ("name:", name)
                #这里我们只做出打印，并不干别的
```

**注：这些函数都包含在我们继承重写的sax类对象中**

```
def characters(self,content):
#内容事件处理，遇到字符串时触发
        if self.CurrentData =='age':
            self.age = content
        elif self.CurrentData == "sex":
            self.sex = content
        elif self.CurrentData == "salary":
            self.salary = content
```

**注：这些函数都包含在我们继承重写的sax类对象中**

```
def endElement(self,tag):
#遇到结束标签时触发
        if self.CurrentData == "age":
        #初始化我们之前要获取的当前标签这个数据对象
            print ("age:",self.age)
        elif self.CurrentData == "sex":
            print ("sex:",self.sex)
        elif self.CurrentData == "salary":
            print ("salary:",self.salary)
        #最后你都得把这个类对象里的CurrentData重新初始化成空
        self.CurrentData = ""
```

# 3 Dom解析XML

# 3.1 Dom解析XML

**DOM(Document Object Model)（Document Object Model）**
**（占用内存大，基于对象的API）**

**与SAX比较，DOM典型的缺点是比较慢，消耗更多的内存，因为DOM会将整个XML数读入内存中，并为树中的第一个节点建立一个对象。使用DOM的好处是你不需要对状态进行追踪，因为每一个节点都知道谁是它的父节点，谁是子节点。但是DOM用起来有些麻烦。**

# 3.1.1 Dom 节点类型

#'ATTRIBUTE_NODE'
#'CDATA_SECTION_NODE'
#'COMMENT_NODE'
#'DOCUMENT_FRAGMENT_NODE'
#'DOCUMENT_NODE'
#'DOCUMENT_TYPE_NODE'
#'ELEMENT_NODE'
#'ENTITY_NODE'
#'ENTITY_REFERENCE_NODE'
#'NOTATION_NODE'
#'PROCESSING_INSTRUCTION_NODE'
#'TEXT_NODE'

**整个文档是一个文档节点**
**每个XML标签是一个元素节点**
**包含在XML元素中的文本是文本节点**
**每一个XML属性是一个属性节点**
**注释属于注释节点**

**http://www.w3school.com.cn/xmldom/dom_nodes.asp**

# 3.2 Dom 读API介绍

导入模块：

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

读xml：

加载xml文件

```
mydom = minidom.parse(file_name)
```

获取xml文档对象

```
root = doc.documentElement
```

获取节点对象集合

```
root.getElementsByTagName(tag_name)
```

获取节点属性

```
node.getAttribute(AttributeName)
```

返回子节点列表

```
node.childNodes
```

返回子节点的值

```
node.childNodes[index].nodeValue
```

访问第一个节点

```
node.firstChild
```

# 3.3 Dom 读xml代码

1、导入模块

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

2、打开xml文件

```
mydom = minidom.parse('read.xml')
```

3、得到xml文档对象

```
root = mydom.documentElement
```

4、从root节点中获取到所有people节点

```
people_nodes = root.getElementsByTagName("people")
```

```
# -*- coding:utf-8 -*-
def parsexml(node):
        for people_node in people_nodes:
            name = people_node.getAttribute("name")
            #获取到people节点的属性
            #people_node获取到的就是每一个people节点
            nodes_age = people_node.getElementsByTagName("age")
            nodes_sex = people_node.getElementsByTagName("sex")
            nodes_salary = people_node.getElementsByTagName("salary")
            #getElementsByTagName 获取到的是一个 结果集 ，所以我们要取出我们第一个

            age = nodes_age[0].childNodes[0].nodeValue
            sex = nodes_sex[0].childNodes[0].nodeValue
            salary = nodes_salary[0].childNodes[0].nodeValue
            #nodes_age[0] 首先我们把获取到的age节点中的第一个age节点取出来
            #childNodes[0] 因为我们每一个age都可能对应多个子节点，
            #所以我们还继续得使用childNodes[0]取出第一个文本节点，其实我们也只有一个文本节点
            #nodeValue 节点的值
            print("----------------")
            print("name:",name)
            print("age:",age)
            print("sex:",sex)
            print("salary:",salary)
parsexml(people_nodes)
```

# 3.4 Dom 写API介绍

写xml：

生成XML节点

doc.createElement(Node_name)

给节点添加属性值

node.setAttribute(att_name,att_value)

节点的内容

doc.createTextNode(text)

添加到指定的节点下面

node.appendChild(Node_name)

把内存中的xml写入到文件中

doc.writexml(file)

打开一个文件，把doc优雅的写入到文件中

xmlfile.write(doc.toprettyxml())

# 3.5 Dom 写xml代码

1、导入模块

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

2、创建一个Dom对象

```
doc = minidom.Document()
```

3、创建一个根节点

```
person = doc.createElement("person")
doc.appendChild(person)
```

```
def addNode(Node_Dict):
        people = doc.createElement("people")
        #createElement 创造一个节点

        people.setAttribute("name",Node_Dict["name"])
        #setAttribute 用来设置节点属性

        age = doc.createElement("age")
        age.appendChild(doc.createTextNode(Node_Dict["age"]))
        people.appendChild(age)
        #把这个节点加到了people这个节点下面
        #继续创建一个age节点
        #appendChild添加子标签，并且是创建了一个文本内容

        sex = doc.createElement("sex")
        sex.appendChild(doc.createTextNode(Node_Dict["sex"]))
        people.appendChild(sex)

        salary = doc.createElement("salary")
        salary.appendChild(doc.createTextNode(Node_Dict["salary"]))
        people.appendChild(salary)

        person.appendChild(people)
        #把这个节点追加到根节点下面
```

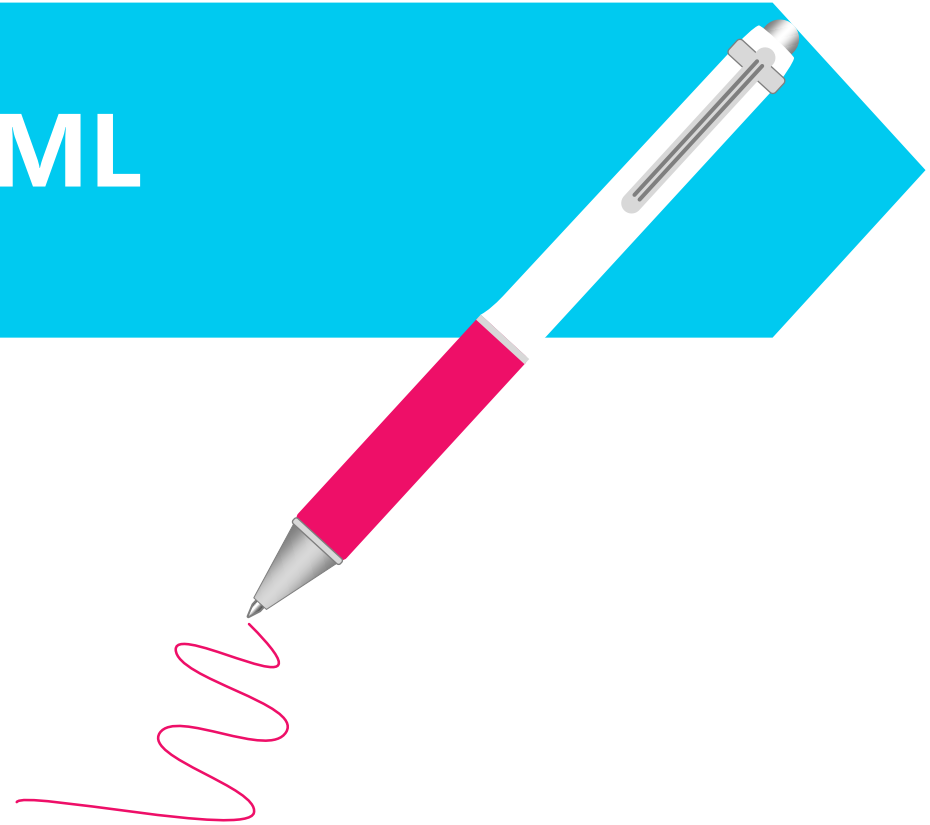
# 3.6 SAX与DOM解析比较

SAX可以快速扫描一个大型的XML文档，当它找到查询标准时就会立即停止，然后再处理之。

DOM是把XML全部加载到内存中建立一棵树之后再进行处理。所以DOM不适合处理大型的XML【会产生内存的急剧膨胀】。

DOM的弱项就是SAX的强项，SAX不必把全部的xml都加载到内存中。但是SAX的缺点也很明显，它只能对文件顺序解析一遍，不支持对文件的随意存取。SAX也仅仅能够读取文件的内容，并不能修改内容。DOM可以随意修改文件树，从而起到修改xml文件的作用

# 4 Etree解析XML



# 4.1 etree解析XML

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。

ElementTree生来就是为了处理XML，它在Python标准库中有两种实现：一种是纯Python实现的，如xml.etree.ElementTree，另一种是速度快一点的xml.etree.cElementTree。

注意：尽量使用C语言实现的那种，因为它速度更快，而且消耗的内存更少。

这是一个让Python不同的库使用相同API的一个比较常用的办法，而从Python 3.3开始ElementTree模块会自动寻找可用的C库来加快速度，所以只需要 import xml.etree.ElementTree 就可以了。

# 4.1.1 etree导入数据

**我们有多种方法导入数据。**

**从硬盘文件导入：**

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
root = tree.getroot()
```

**通过字符串导入：**

```
root = ET.fromstring(country_data_as_string)
```

**fromstring()**
**解析XML时直接将字符串转换为一个　Element，解析树的根节点。其他的解析函数会建立ElementTree。一个Element有一个tag以及一些列属性（保存在dictionary中）**

# 4.2 etree 读API介绍

读xml：（element 是一个元素，node是一部分节点，囊括一部分元素）

获取节点标签值

```
node.tag
```

获取节点属性和属性值的字典

```
node.attrib
```

获取node下的子节点，迭代遍历子树

```
for child in node:
    node.iter(node_name)
```

用来查找当前node中的属于某个tag的所有node

```
node.findall(node_name)
```

查找第一个tag

```
node.find(node_name)
```

获取到node的文本节点

```
node.text()
```

获取attr_name对应的属性值

```
node.get(attr_name)
```

# 4.2.1 etree 读XML代码

**导入模块：**

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

**在内存构建内存树：**

```
tree = ET.parse("ETread.xml")
```

**获取root节点：**

```
root = tree.getroot()
```

**遍历子节点：**

```
for child in root:
print(child.tag,child.attrib)
```

```
# -*- coding:utf-8 -*-
import xml.etree.ElementTree as ET
#导入模块

tree = ET.parse("ETread.xml")
root = tree.getroot()
print(root.tag)
#打印出root的标签值

#遍历子节点
for people in root.findall("people"):
        print("-------------")
        print(people.attrib)
        #打印节点属性和属性值的字典

        #print("name:",people.get("name"))
        #打印属性中name对应的值

        age = people.find("age")
        #find和findall的区别，只找出第一个符合条件的节点
        sex = people.find("sex")
        salary = people.find("salary")
        print("age:",age.text)
        print("sex:",sex.text)
        print("salary:",salary.text)
```

# 4.3 etree 写API

写xml：

创建内存树：

ET.ElementTree()

创建节点：

Element(node_name)

设置为根节点：

node._setroot

添加子节点及其属性，返回一个子节点对象：

node = SubElement( father_node,son_node_name,{ attr_name:attr_value } )

追加子节点到父节点上：

root.append(node)

```
# -*- coding:utf-8 -*-
import xml.etree.ElementTree as ET

xmltree =  ET.ElementTree()
person = ET.Element("person")
#创建节点

xmltree._setroot(person)
#在xml对象中设置根节点

people = ET.SubElement(person,'people',{"name":"Jack"})
#给根节点person创建子节点people

age = ET.Element("age")
#创建age节点
people.append(age)
age.text = "18"
#挂到people节点下面，设置文本节点内容

#这是我们另外一种创建节点的方式，先把节点搞出来
#然后可以通过 node.text 或者 node.set 添加文本和属性
sex = ET.SubElement(people,'sex')
sex.text = "male"
salary = ET.SubElement(people,'salary')
salary.text = "13000"
```

**格式化节点样式函数：**

```
def  indent(elem, level=0):
    i = "\n" + level*"  "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + "  "
        for e in elem:
            indent(e, level+1)
        if not e.tail or not e.tail.strip():
            e.tail = i
    if level and (not elem.tail or not elem.tail.strip()):
        elem.tail = i
    return elem

ET.dump(indent(person))
#格式化节点样式，如果你不选择格式化，可能你的XML中格式为一整行，没有换行符。
xmltree.write("ETwrite.xml")
```

# 4.4 etree 修改XML API

**修改文本：**

Element.text

**添加或者修改节点的属性：**

Element.set(key,value)

**添加节点：**

Element.append(node_name)

**删除节点：**

Element.remove(node_name)

# 4.4.1 etree 修改XML代码

```
# -*- coding:utf-8 -*-
import xml.etree.ElementTree as ET
tree = ET.parse("ETChange.xml")
root = tree.getroot()
for people in root.findall("people"):
        print("-----Before-----")
        print(people.attrib)
        print("-----after-----")
        people.set("update","yes")#添加属性
        print(people.attrib)

        age = people.find("age")
        sex = people.find("sex")
        salary = people.find("salary")

        print("-----Before-----")
        print("age:",age.text)
        print("sex:",sex.text)
        print("salary:",salary.text)
        print("-----after-------")
        age.text = str(eval(age.text)+2)#修改年龄
        print("age:",age.text)
        print("sex:",sex.text)
        print("salary:",salary.text)
tree.write("ETChange.xml")
```

# Thank you for listening

赤水教育Python学院

讲师QQ:3123017211