XPath的学习

# XPath简单介绍：

**XPath 是一门在 XML 文档中查找信息的语言。方法：使用路径表达式。X： 指XML；Path：指路径表达式。**

XPath 使用**路径表达式**在 XML 文档中进行导航；XPath 可用来在 XML 文档中对**元素和属性**进行遍历。

**XPath** 使用**路径表达式**来选取 XML 文档中的节点或节点集。节点是通过沿着**路径 (path) 或者步 (steps)** 来选取的。

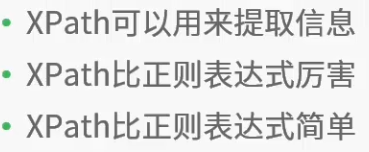
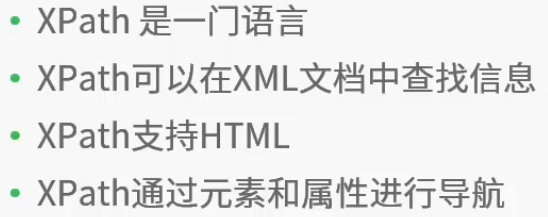
XPath 包含一个**标准函数库**；XPath 是 XSLT 中的主要元素；XPath 是一个 **W3C 标准**。

**XPath** 是 **W3C XSLT 标准**的主要元素，并且 **XQuery 和 XPointer 都构建于 XPath 表达之上**。

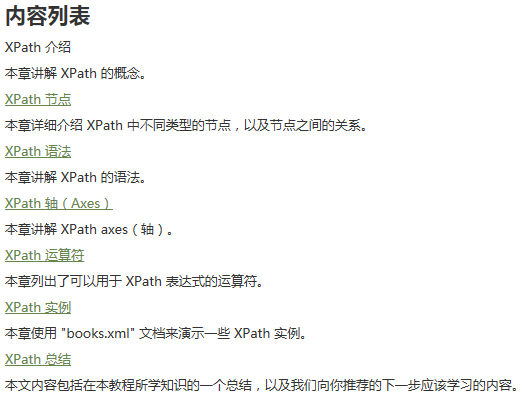
因此，**对 XPath 的理解是很多高级 XML 应用的基础**。



xpath的作用就是两个字“定位”，运用各种方法进行快速准确的定位，推荐两个非常有用的的firefox工具：firebug和xpath checker。



# XPath学习内容列表



# XPath的获取方法

### 手动编写

### 浏览器目前都支持自动生成标签的XPath路径。

# XPath中基本概念的介绍

## XPath路径表达式

XPath 使用**路径表达式**来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

## XPath标准函数

**XPath 含有超过 100 个内建的函数**。这些函数用于**字符串值、数值、日期和时间比较、节点和 QName 处理、序列处理、逻辑值**等等。

## XPath在XSLT中的使用

XPath 是 XSLT 标准中的主要元素。如果没有 XPath 方面的知识，您就无法创建 XSLT 文档。

XQuery 和 XPointer 均构建于 XPath 表达式之上。XQuery 1.0 和 XPath 2.0 共享相同的数据模型，并支持相同的函数和运算符。

## XPath 是 W3C 标准

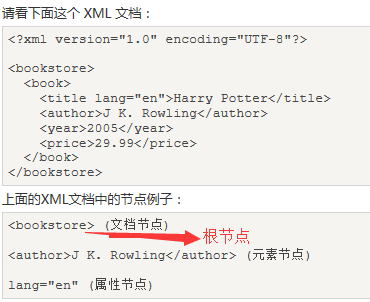
XPath 于 1999 年 11 月 16 日 成为 W3C 标准。

XPath 被设计为供 XSLT、XPointer 以及其他 XML 解析软件使用。

# XPath术语

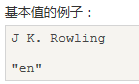
## 节点node

在 XPath 中，有七种类型的节点：**元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点**。XML 文档是被作为**节点树**来对待的。树的根被称为**文档节点或者根节点**。



## 基本值（或称原子值，Atomic value）

基本值是无父或无子的节点。



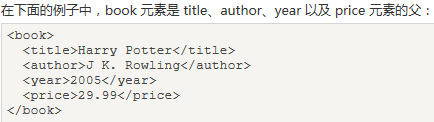
## 项目（Item）

项目是基本值或者节点。

# 节点关系

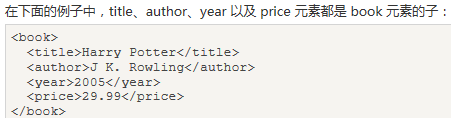
## 父（Parent）

每个元素以及属性都有一个父。（除根节点）



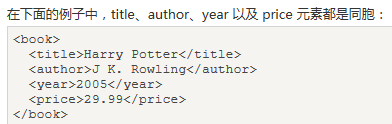
## 子（Child）

元素节点可有**零个、一个或多个子**。



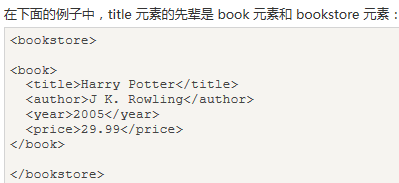
## 同胞（Sibling）

拥有相同的父的节点。



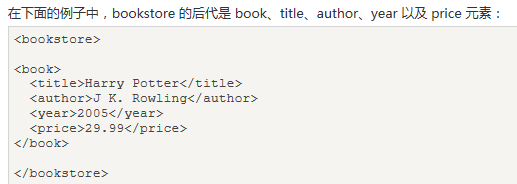
## 先辈（Ancestor）

某节点的父、父的父，等等。

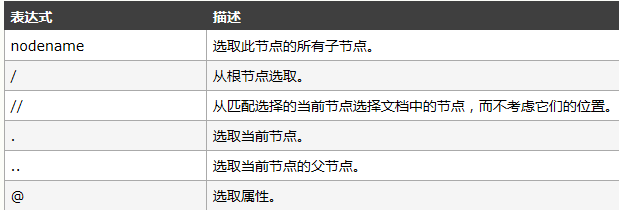


## 后代（Descendant）

某个节点的**子，子的子**，等等。



# Xpath的语法（特殊符号）



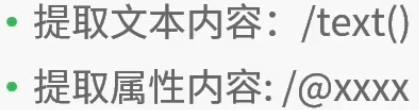
一定注意: 如果是某个节点下的子节点，利用nodename获取，不要利用//nodename，后者会扩大范围,例如body.xpath(‘div’) 与 body.xpath(‘//div’)

实例：



**提取文本内容： /text() 注意：一定要加括号**

**提取属性内容：/@xxxx**



# 谓语（Predicates）

**谓语**用来查找**某个特定的节点或者包含某个指定的值的节点**。

**谓语被嵌在方括号中**。

实例：带有谓语的一些路径表达式。



# 利用通配符选取未知节点

**XPath** 使用路径表达式在 **XML 文档**中选取节点。节点是通过沿着路径或者 step 来选取的。

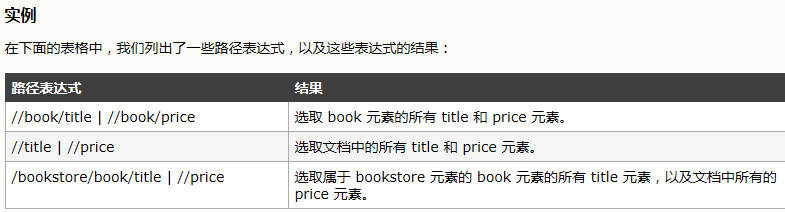
**XPath 通配符**可用来选取未知的 **XML 元素**。





# 选取若干路径

通过在**路径表达式**中使用“**|**”运算符，您可以选取若干个路径。



# XPath 轴（Axes）

**轴（Axes）：相对于当前节点的节点集**。



# Xpath运算符

XPath 表达式可返回节点集、字符串、逻辑值以及数字。

下面列出了可用在 **XPath 表达式**中的运算符：

## | 或

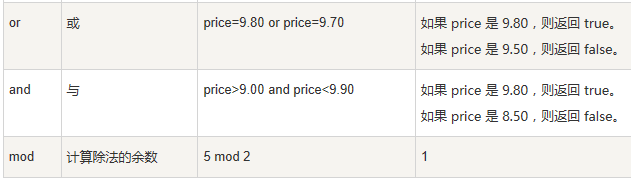
## 加减乘除、求余 + - \* div mod

注意：除为div ，因为/ 已经有特殊意义。

## 关系运算符： =、!=、<、>、>=、<=

## 逻辑运算符：or and



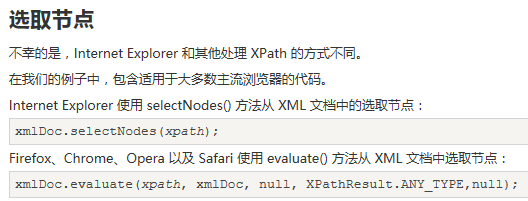


# XPath实例

books.xml:

<?xml version="1.0" encoding="UTF-8"?>  
  
< 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">XQuery Kick Start</title>  
 < author>James McGovern</author>  
 < author>Per Bothner</author>  
 < author>Kurt Cagle</author>  
 < author>James Linn</author>  
 < author>Vaidyanathan Nagarajan</author>  
 < year>2003</year>  
 < price>49.99</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>





### 选取所有 title: /bookstore/book/title

### 选取第一个 book 的 title:/bookstore/book[1]/title

这里有一个问题。上面的例子在 IE 和其他浏览器中输出不同的结果。

IE5 以及更高版本将 [0] 视为第一个节点，而根据 W3C 的标准，应该是 [1]。

一种解决方法！

为了**解决 IE5+ 中 [0] 和 [1] 的问题**，可以为 XPath 设置语言选择（SelectionLanguage）。

下面的例子选取 bookstore 元素下面的第一个 book 节点的 title：

xml.setProperty("SelectionLanguage","XPath");

xml.selectNodes("/bookstore/book[1]/title");

### 选取所有价格

下面的例子选取 price 节点中的所有文本：

/bookstore/book/price/text()

### 选取价格高于 35 的 price 节点

下面的例子选取价格高于 35 的所有 price 节点：

/bookstore/book[price>35]/price

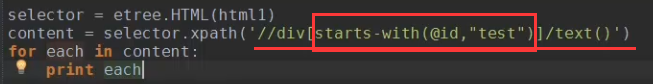
### 选取价格高于 35 的 title 节点

下面的例子选取价格高于 35 的所有 title 节点：

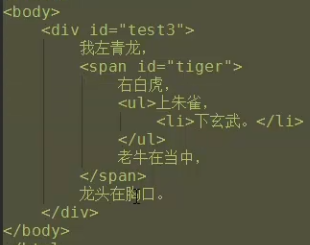
/bookstore/book[price>35]/title

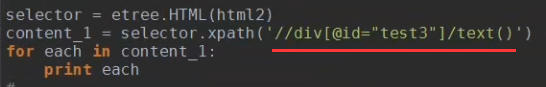
### 特殊情况的提取：starts-with

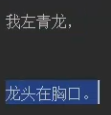




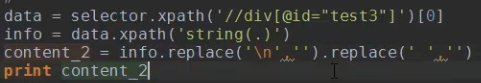
### 标签套标签 string(.)







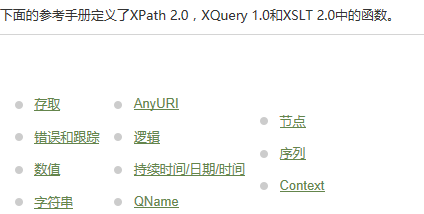
**解决办法**：利用 **string(.) 来提取内部所有文本及内部所有的标签对应的文本。**





# XPath、Xquery以及XSLT 函数 链接

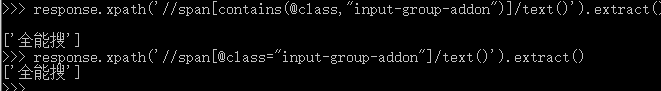
<http://www.runoob.com/xpath/xpath-functions.html>



# 常用函数

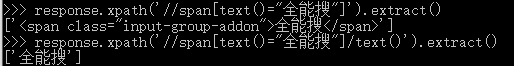
## contains(@attr,”value”):包含某个属性

和字面意思一样就是包含，例如：//div[contains(@class,'xxx')]



## text():通过限制文本搜取，或获取文本内容



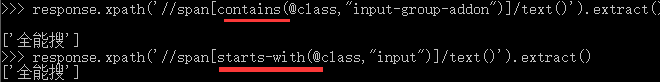


**contains(text(),'xxx')**

## starts-with(@attr,”xxx”)

注意：**在xpath中没有ends-with**

和字面意思一样就是以某某开头，例如：//input[starts-with(@class,'xxx')]



## not

就是否定的意思，比如找一个id不为123的input：input[not[id='123']]

又如找一个文本中不包含xxx字段的**span：//span[not(contains(text(),'xxx'))]**

## position()=num 与 last() 、last()-num ：索引（从1开始）

<div id="positions">

<input>

<span>test position()1</span>

<span>test position()2</span>

<span>test position()3</span>

<span>test position()4</span>

<span>test position()5</span>

</input>

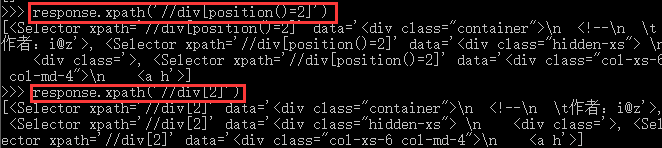
</div>

获取第一个span，可以是**//div[@id='positions']/span[1]**，也可以是**//div[@id='positions']/span[position()=1]**

**//div[@id='positions']/span[position()>3]**

就是定位了test position()4和test position()5

//div[@id='positions']/span[position()<3]就是定位了test position()1和test position()2



## substring,substring-before,substing-after（使用比较少）

sbustring(str,start-position,length) 比如html：

<div id="xxx">

<span name="?-xxxxx-09">text</span>

</div>

定位上面html中span：//div[@id='xxx']/span[substring(@name,3,5)='xxxxx']

substring-before的用法，比如html

<div id="xxx">

<span class="spanclass1-789">text</span>

</div>

定位上面html中span：//div[@id="xxx"]/span[**sbustring-before(@class,"-")="spanclass1"**]

substring-after的用法，比如html

<div id="xxx">

<span class="789-spanclass2">text</span>

</div>

定位上面html中span：//div[@id="xxx"]/span[sbustring-after(@class,"-")="spanclass22"]

## following-sibling 与preceding-sibling

### following-sibling

**following-sibling即为“选择当前节点之后的所有同级节点”**，那么没有加上“sibling”关键字的，搜索的就是之上/之下的所有节点，忽略同级概念，例如：

<div>

<input id="123">

<input></input>

</div>

要定位第二个input：//input[@id='123']/**following-sibling::input**

### preceding-sibling

**preceding-sibling的解释是“选取当前节点之前的所有同级节点”**，那么没有加上“sibling”关键字的，搜索的就是之上/之下的所有节点，忽略同级概念， preceding-sibling和following-sibling是刚好相反的

<div>

<span>text</span>

<input id="123">

</div>

要定位第二个input：//input[@id='123']/**preceding-sibling::span**

## Axes轴

**Axes 轴：使用的时候注意加上::**

parent 父节点

ancestor 祖先节点，包括父节点，一层一层向上

descendant 所有子孙节点找，不管什么位置，简写//，就是xpath中出现//的情况。。//div[@class="xxx"]//input

follwing-sibling 当前元素后面的兄弟姐妹

preceding-sibling 当前元素前面的兄弟姐妹

following 当前元素后面所有元素，一直到</html>

preceding  当前元素之前所有元素，一直到<html>

ancestor-or-self

descendant-or-self

# text()、string()、data()的使用和区别

## text()与string()的区别

**摘要**： 在XPath中，经常使用**text()和string()**，而我一般都是想到哪个用哪个，究竟他们之间有什么不同，没有在意过。

**本质区别**：text()是一个node test，而**string()是一个函数**，**data()是一个函数且可以保留数据类型**。此外，还有点号（.）表示当前节点。



示例：



XML例子：

<book>

<author>Tom <em>John</em> cat</author>

<pricing>

<price>20</price>

<discount>0.8</discount>

</pricing>

</book>

### text()

经常在XPath表达式的最后看到**text()**，它仅仅返回**所指元素的文本内容**。

**let $x := book/author/text()**

**return $x**

返回的结果是**Tom cat**，其中的John不属于author直接的节点内容。

### string()

**string()函数**会得到所指元素的所有节点文本内容，这些文本讲会被拼接成一个字符串。

**let $x := book/author/string()**

**return $x**

返回的内容是”**Tom John cat**”

### data()

大多数时候，**data()函数**和**string()函数**通用，而且不建议经常使用data()函数，有数据表明，该函数会影响XPath的性能。

**let $x := book/pricing/string()**

**return $x**

返回的是200.8

**let $x := book/pricing/data()**

**return $x**

这样将返回分开的20和0.8，他们的类型并不是字符串而是xs:anyAtomicType，于是就可以使用数学函数做一定操作。

let $x := book/pricing/price/data()

let $y := book/pricing/discount/data()

return $x\*$y

比如上面这个例子，就只能使用**data()**，**不能使用text()或 string()**，因为XPath不支持字符串做数学运算。

### 总结

**text()不是函数**，XML结构的细微变化，可能会使得结果与预期不符，应该尽量少用，**data()作为特殊用途的函数，可能会出现性能问题**，如无特殊需要尽量不用，**string()函数可以满足大部分的需求**。

## string()存在的问题

使用的时候，这样是合法的：

nameList = infoList.xpath('div[@class="hd"]/a/span[1]').xpath('string()').extract()

但是，这样就报错：

nameList = infoList.xpath('div[@class="hd"]/a/span[1]/string()').extract()

而 **text()函数两种形式都是可以的**。

nameList = infoList.xpath('div[@class="hd"]/a/span[1]/text()').extract()

nameList = infoList.xpath('div[@class="hd"]/a/span[1]').xpath('text()').extract()

# 如何在Python中单独使用XPath？

**对XPath的支持存在于lxml库中**。具体使用方法：



### 第一步：安装lxml库

pip install lxml

### 在代码中导入etree

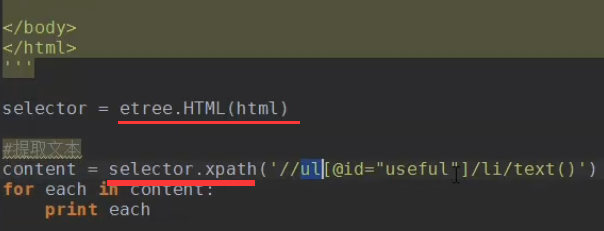
**from lxml import etree**

### 将源代码转化为Selector对象，进而利用Selector的xpath方法

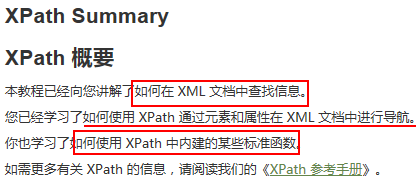
**Selector = etree.HTML(‘htmlcode’)**

**Selector.xpath(‘\*\*\*\*\*’)**

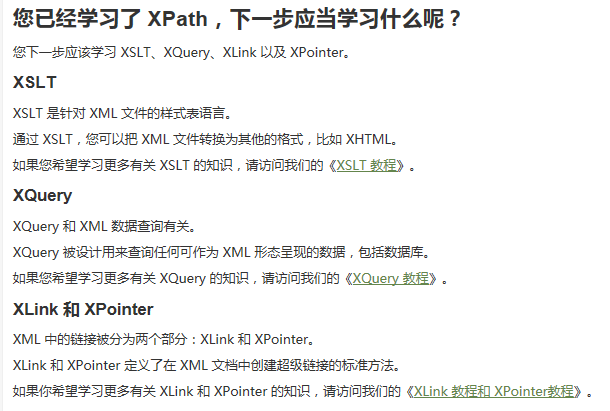
示例：



# XPath总结Summary



<http://www.runoob.com/xpath/xpath-summary.html>



# XPath路径表达式练习

一般在检验XPath是否正确的时候，通常都是利用

**scrapy shell** [**http://www.\*\*\*.\*\*\*\*\***](http://www.***.*****) 命令检验。

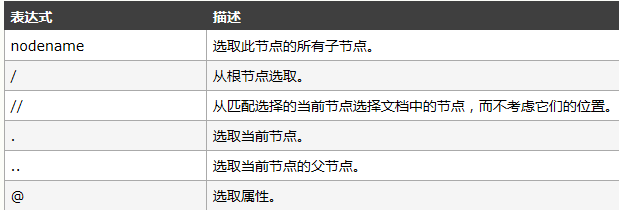


选择网址：<http://www.51voa.com/VOA_Videos/doctors-worry-about-post-antibiotic-future-74235.html>

## scrapy shell <http://www.51voa.com/VOA_Videos/doctors-worry-about-post-antibiotic-future-74235.html>

**进入shell**

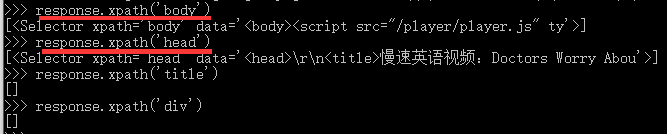
## 练习使用XPath

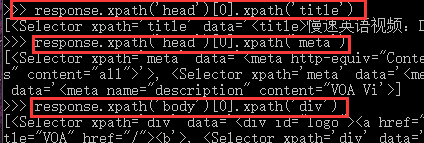


### nodename：选取当前节点的所有字节点（只是儿子节点）中的nodename节点

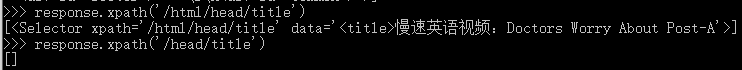
直接使用nodename，这个node必须是当前节点的子节点才可以选中。

这个例子中，response对象默认的就是跟节点html，所以，只有当nodename为head或body时，才可以选中节点，对于title、div等，无法选中。

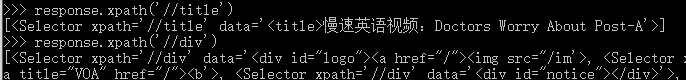


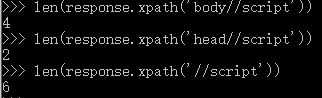


### / ：从根节点开始选取（绝对路径）



### // : 从当前节点开始寻找其内部的所有的后代节点





由于当前节点是html，且body和head都是html的儿子节点，所以可以利用body//script、head//script寻找。

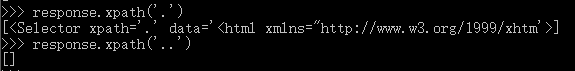
### 注意/与// 的最大区别：

/ 强调从根出发，//强调从当前节点出发。

/ 只取出儿子节点，// 取出所有的后代节点。所以一般情况下，使用的是//.

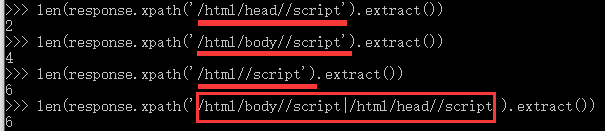
### . 当前节点 .. 当前节点的父节点

Response的当前节点就是根节点html，其没有父节点。





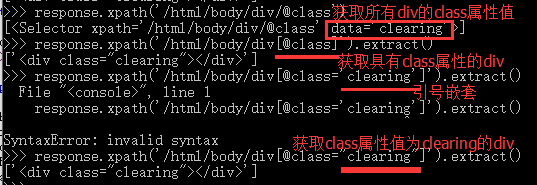
### 利用 | 连接多个选择



### @ 获取属性（）

注意：**与谓语不同，谓语部分是利用方括号括起来，表示的是一种限制条件，而这里不需要方括号，目的就是获取其属性值**。

response.xpath('/html/head/script/@src').extract()

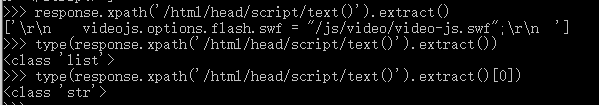




### 利用text()函数获取文本内容

/html/head/script/text()

返回的是一个字符串的list类型。





限制条件多个，用and或or连接。

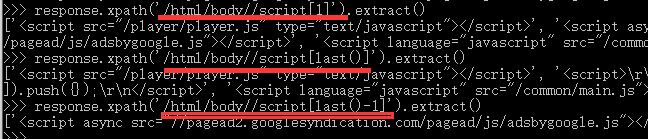
### 利用通配符获取未知节点

\*匹配所有的节点

@\* 至少有一个属性



### 谓语：限制条件



等等。