soup = BeautifulSoup(string, string转换成BeautifulSoup对象 'html.parser') title标签 soup.title 'title' soup.title.name title标签内容 soup.title.string 子孙节点在内的所有内容 soup.title.text 子孙节点在内的所有内容 soup.title.get_text() 'head' soup.title.parent.name 字典 soup.a.attrs 浏览数据 是一个列表 p标签class属性 soup.p['class'] 属性 soup.p.get('class') tag是否有class属性 tag.has_attr('class') 根据标签名选取第一个标签 soup.a 查找 根据标签名选取所有的标签 soup.find_all('a') 根据属性选取第一个标签 soup.find(id='link') .name 获取 没有会KeyError 使用tag.get('class') tag['class'] 属性 soup.标签名 'class'返回列表,非多值属性为 字符串,如'id' 增删改 与字典一样 获取 .string .string.replace_with('替换的文 替换 本') Tag中的字符串 BeautifulSoup对象 NavigableString BeautifulSoup之外使 用,转换成Unicode字 符串,以节省内存 unicode(tag.string) 文档的全部内容,不是Tag对象 BeautifulSoup 搜索 Comment .标签名 Tag名字 获得当前名字的第一个tag 链式调用 soup.body.b 列表 直接子节点包含孙节点的内容 tag.contents list_iterator迭代器 tag.children 直接子节点 如果只有一个子节点,可用 tag.string输出内容 直接子节点不包括本节点里面的 文本内容 tag有多个子节点,.string结果为 子孙节点 generator生成器 包含子节点和孙节点 tag.descendants tag只包含一个子节点 使用tag.string len(tag.contents) == 1 文本内容 生成器 使用tag.strings tag包含多个子节点 生成器,去除空格、空行 tag.stripped_strings 遍历文档树 获取所有文本内容包括子孙节点 get_text(strip=True) tag.parent 顶层节点,如html,其父节点是 直接父节点 BeautifulSoup对象 BeautifulSoup.parent是None for parent in tag.parents: if parent is None: print(parent) print(parent.name) 所有父节点 tag.parents 祖先节点 经常返回',\n' tag.next_sibling 最顶层tag没有,则为None tag.previous_sibling 兄弟节点 tag.next_siblings 生成器 tag.previous_siblings 字符串 正则表达式 列表 True name参数 查找名为name的tag 函数名 字符串、正则表达式、列表、 True 如find_all(id='link') keyword参数 find_all(name, attrs, 字典 recursive, string, **kwargs) 字符串、正则表达式、函数名、 如soup.find_all("a", True class_="sister") 按CSS搜索 soup.find_all("p", class_="body 多值属性可完全匹配 strikeout") 用来匹配文档中字符串的内容 string参数 查找子孙节点 limit参数 限制结果的最大数量 是否递归查询子孙节点 默认查询为True 只想匹配直接子节点,设置 recursive参数 recursive=False find_all()返回列表,find()返回结 find(name, attrs, recursive, string, **kwargs) find_all()找不到返回[],find()返 回None 搜索文档树 find_parents() 查找父节点 find_parent() find_next_siblings() find_next_sibling() 查找兄弟节点 find_previous_siblings() find_previous_sibling()

tag.select()

tag.select_one()

tag.name = 新值

tag['属性'] = 新值

tag.string = 新值

1、new_string构造注释;

1、new_tag构造标签

2、append添加

2 append

tag.append('新增字符串')

insert与append相比,不会把元素插

引处插入,与列表的insert()类似

tag.clear()

入到父节点,只会在本tag里指定的索

insert_before()

insert_after()

del tag['属性']

添加

注释

标签

insert插入

当前节点/文本前后插入

当前tag从文档中移除,并返回当

replace_with() 方法返回被替代的

wrap()对指定的tag元素进行包装

unwrap()移除知道tag标签(包括属

从文档中移除,且不返回tag

当前tag文本内容

前tag的内容

tag或文本节点

性),内容不变

返回第一个

tag下嵌套tag,会抹掉里面的tag

字符串可以构造成

from bs4 import Comment

tag.append(new_comment)

new_tag = soup.new_tag("标签

new_comment = soup.new_string("注释内容",

NavigableString

Comment)

tag.append(new_tag)

ex = tag.extract()

tag.replace_with(new_tag)

tag.wrap(soup.new_tag("div"))

tag.decompose()

old_tag =

tag.append("Hello")

tag.append(new_string)

标签里内嵌新标签

s = ex.string.extract()

there")

ex可以继续调用extract()

new_string = NavigableString("

CSS选择器

修改name

修改属性

文本内容

插入

移除

替换

解包

外包装

修改文档树

找到字符串标签

tag.has_attr('id')

def not_lacie(href):

所有标签

通过search()来匹配

def has_class_but_no_id(tag):

return href and not

soup.find_all(href=not_lacie)

HTML5中有些属性中间有"-",如

data-foo,不能直接传递带有"-"

的属性名,使用attrs代替

字符串、正则表达式、列表、

只查询tag的直接子节点,过

滤掉孙节点,如果查不到,返

True

回空列表'[]'

re.compile("lacie").search(href)

return tag.has_attr('class') and not

soup.find_all(has_class_but_no_id)

tag

id=True, 查找所有拥有id属性的

len(css_class) == 6

"value"})

return css_class is not None and

soup.find_all(class_=has_six_characters)

def has_six_characters(css_class):

soup.find_all(attrs={"data-foo":

对列表每一元素都进行匹配

选择 id="firstname" 的所有元素。 #firstname 选择所有元素。 element 选择所有 元素。 选择所有 <div> 元素和所有 元素。 div,p element,element 选择 <div> 元素内部的所有 元素。 element element 选择父元素为 <div> 元素的所有 元素。 div>p element>element 选择紧接在 <div> 元素之后的所有 元素。 div+p element+element 选择带有 target 属性所有元素。 [attribute] [target] 选择 target="_blank" 的所有元素。 [attribute=value] [target=_blank] 选择 title 属性包含单词 "flower" 的所有元素。 [title~=flower] [attribute~=value] [lang|=en] 选择 lang 属性值以 "en" 开头的所有元素。 [attribute|=value] 选择所有未被访问的链接。 a:link 选择所有已被访问的链接。 a:visited 选择活动链接。 a:active 选择鼠标指针位于其上的链接。 :hover a:hover 选择获得焦点的 input 元素。 :focus input:focus :first-letter 选择每个 元素的首字母。 p:first-letter :first-line 选择每个 元素的首行。 p:first-line 选择属于父元素的第一个子元素的每个 元素。 :first-child p:first-child 在每个 元素的内容之前插入内容。 :after 在每个 元素的内容之后插入内容。 p:after 选择带有以 "it" 开头的 lang 属性值的每个 元素。 :lang(*language*) p:lang(it) 选择前面有 元素的每个 元素。 element1~element2 选择其 src 属性值以 "https" 开头的每个 <a> 元素。 a[src^="https"] [attribute^=value] a[src\$=".pdf"] 选择其 src 属性以 ".pdf" 结尾的所有 <a> 元素。 [attribute\$=value] 选择其 src 属性中包含 "abc" 子串的每个 <a> 元素。 a[src*="abc"] [attribute*=value] p:first-of-type 选择属于其父元素的首个 元素的每个 元素。 :first-of-type 选择属于其父元素的最后 元素的每个 元素。 :last-of-type p:last-of-type 选择属于其父元素唯一的 元素的每个 元素。 :only-of-type p:only-of-type :only-child 选择属于其父元素的唯一子元素的每个 元素。 p:only-child :nth-child(n) 选择属于其父元素的第二个子元素的每个 元素。 p:nth-child(2) 同上,从最后一个子元素开始计数。 :nth-last-child(n) p:nth-last-child(2) 选择属于其父元素第二个 元素的每个 元素。 :nth-of-type(n) p:nth-of-type(2) 同上,但是从最后一个子元素开始计数。 p:nth-last-of-type(2) 选择属于其父元素最后一个子元素每个 元素。 :last-child p:last-child 选择文档的根元素。 选择没有子元素的每个 元素(包括文本节点)。 :empty p:empty :target #news:target 选择当前活动的 #news 元素。 选择每个启用的 <input> 元素。 :enabled input:enabled 选择每个禁用的 <input> 元素 :disabled input:disabled :checked 选择每个被选中的 <input> 元素。 input:checked

选择非 元素的每个元素。

选择被用户选取的元素部分。

bs4

例子描述

选择 class="intro" 的所有元素。

CSS

<page-header> 由 Xnip 截图

例子

CSS选择器

:not(selector)

::selection

:not(p)

::selection