大家在使用selenium元素定位的时候，通常更多使用的是XPATH，css定位方式用得比较少

但有时候css定位方式还是有一些优势的，

优势1：一般情况下定位速度要比XPATH快

优势2：语法要比XPATH更简洁

下面简要介绍一下css元素选择器的语法

**常见语法**

|  |  |
| --- | --- |
| \* | 通用元素选择器，匹配任何元素 |
| E | 标签选择器，匹配所有使用E标签的元素 |
| .info | class选择器，匹配所有class属性中包含info的元素 |
| #footer | id选择器，匹配所有id属性等于footer的元素 |
| E,F | 多元素选择器，同时匹配所有E元素或F元素，E和F之间用逗号分隔 |
| E F | 后代元素选择器，匹配所有属于E元素后代的F元素，E和F之间用空格分隔 |
| E > F | 子元素选择器，匹配所有E元素的子元素F |
| E + F | 毗邻元素选择器，匹配紧随E元素之后的同级元素F （只匹配第一个） |
| E ~ F | 同级元素选择器，匹配所有在E元素之后的同级F元素 |
| E[att='val'] | 属性att的值为val的E元素 （区分大小写） |
| E[att^='val'] | 属性att的值以val开头的E元素 （区分大小写） |
| E[att$='val'] | 属性att的值以val结尾的E元素 （区分大小写） |
| E[att\*='val'] | 属性att的值包含val的E元素 （区分大小写） |
| E[att1='v1'][att2\*='v2'] | 属性att1的值为v1，att2的值包含v2 （区分大小写） |

例如这样一段html代码的网页

<div class="formdiv">

<form name="fnfn">

<input name="username" type="text"></input>

<input name="password" type="text"></input>

<input name="continue" type="button"></input>

<input name="cancel" type="button"></input>

<input value="SYS123456" name="vid" type="text">

<input value="ks10cf6d6" name="cid" type="text">

</form>

<div class="subdiv">

<ul id="recordlist">

<p>Heading</p>

<li>Cat</li>

<li>Dog</li>

<li>Car</li>

<li>Goat</li>

</ul>

</div>

</div>

**匹配示例：**

|  |  |
| --- | --- |
| locator | 匹配 |
| css=div  css=div.formdiv | <div class="formdiv"> |
| css=#recordlist  css=ul#recordlist | <ul id="recordlist"> |
| css=div.subdiv p  css=div.subdiv > ul > p  css=ul#recordlist>p | <p>Heading</p> |
| css=form + div | <div class="subdiv"> |
| css=p + li  css=p ~ li:nth-child (1) | 二者定位到的都是 <li>Cat</li>  但是storeCssCount的时候，前者得到1，后者得到4 |
| css=form > input[name=username] | <input name="username"> |
| css=input[name$=id][value^=SYS] | <input value="SYS123456" name="vid" type="hidden"> |
| css=input:not([name$=id][value^=SYS]) | <input name="username" type="text"></input> |

**css中的结构性定位**

结构性定位就是根据元素的父子、同级中位置来定位，css3标准中有定义一些结构性定位伪类如nth-of-type，nth-child，但是使用起来语法很不好理解，这里就不做介绍了。

Selenium中则是采用了来自Sizzle的css3定位扩展，它的语法更加灵活易懂

Sizzle Css3的结构性定位语法

|  |  |
| --- | --- |
| E:nth-child (n) | 在其父元素中的E子元素集合中排在第n+1个的E元素 (第一个n=0) |
| p:nth-of-type(2) | 选择属于其父元素第二个 <p> 元素的每个 <p> 元素。 |
| E:first-child | 在其父元素中的E子元素集合中排在第1个的E元素 |
| p:first-of-type | 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。 |
| E:last-child | 在其父元素中的E子元素集合中排在最后1个的E元素 |
| p:last-of-type | 选择属于其父元素的最后 <p> 元素的每个 <p> 元素。 |
| E:only-child | 父元素的唯一一个子元素且标签为E |
| p:only-of-type | 选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。 |
| E:empty | 不包含任何子元素的E元素，注意，文本节点也被看作子元素 |

**匹配示例：**

**例如还是段的代码**

<div class="subdiv">

<ul id="recordlist">

<p>Heading</p>

<li>Cat</li>

<li>Dog</li>

<li>Car</li>

<li>Goat</li>

</ul>

</div>

|  |  |
| --- | --- |
| locator | 匹配 |
| css=ul > li:nth(0) | <li>Cat</li> |
| css=ul > \*:nth(0)  css=ul > :nth(0) | <p>Heading</p> |
| css=ul > li:first | <li>Cat</li> |
| css=ul > :first | <p>Heading</p> |
| css=ul > \*:last  css=ul > li:last | <li>Goat</li> |
| css=ul > li:even | Cat, Car |
| css=ul > li:odd | Dog, Goat |
| css=ul > :even | <p>Heading</p> |
| css=ul > p:odd | [error] not found |
| css=ul > li:lt(2) | <li>Cat</li> |
| css=ul > li:gt(2) | <li>Goat</li> |
| css=ul > li:only-child  css=ul > :only-child  css=ul > \*:only-child | [error] not found (ul没有only-child) |
| css=div.subdiv > :only-child | <ul id="recordlist">  … … … …  </ul> |

Sizzle Css3还提供一些直接选取form表单元素的伪类

* :input: Finds all input elements (includes textareas, selects, and buttons).
* :text, :checkbox, :file, :password, :submit, :image, :reset, :button: Finds the input element with the specified input type (:button also finds button elements).

下面是一些XPATH和CSS的同义locator比较

|  |  |  |
| --- | --- | --- |
| **定位方式** | **XPath** | **CSS** |
| **标签** | //div | div |
| **By id** | //div[@id='eleid'] | div#eleid |
| **By class** | //div[@class='eleclass'] //div[contains(@class,'eleclass')] | div.eleclass |
| **By 属性** | //div[@title='Move mouse here'] | div[title=Move mouse here] div[title^=Move] div[title$=here] div[title\*=mouse] |
| **定位子元素** | //div[@id='eleid']/\* //div/h1 | div#eleid >\* div#eleid >h1 |
| **定位后代元素** | //div[@id='eleid']//h1 | div h1 |
| **By index** | //li[6] | li:nth(5) |
| **By content** | //a[contains(.,'Issue 1164')] | a:contains(Issue 1164) |
| **根据子元素回溯定位父元素** | //li[a[contains(.,'Issue 1244')]] //\*[./a[contains(.,'Issue 1244')]] //ul[.//a[contains(.,'Issue 1244')]] | li{a:contains(Issue 1244)} ul{a:contains(Issue 1244)} |
| **根据邻近元素定位** | //li[preceding-sibling::li[contains(.,'Issue 1244')]] //ul[preceding-sibling::ul[.//a[contains(.,'Issue 1244')]]] | css=li:contains(Issue 1244) + li css=ul{a:contains(Issue 1244)} ~ ul |

依据后辈节点属性来确定

u"//li[following-sibling::li[./samp[text()='XXXX']]][./samp[text()='YYYYY']]/samp"

先确定后辈节点，再往前找同辈节点

u"//li[./samp[text()='XXXX']]/preceding-sibling::li[./samp[text()='YYYYY']]/samp"