

用户手册

1 AutoMan 简介

AutoMan 是淘宝自动化测试组自行研发的一套界面自动化测试框架。框架的核心是基于界面模型的设计，将“元素查找”和“控件操作”分开。元素查找的方式定义在 PageModel 的 Web 服务器上，在脚本中只说明使用控件的名称和对该控件的操作方式。因此用该框架编写脚本具有上手快、易维护的特点。

1.2 项目中的应用

目前大部分的 web 自动化测试都是应用与回归测试，鲜少有在项目中开展 web 自动化的，其原因就在与选择自动化测试时，我们会考虑：1. 页面设计变化频繁；2. 项目周期足够长；3. 自动化测试脚本可重复使用。而现实世界中尤其是像淘宝这类一直追求敏捷开发为主的公司，其软件项目往往不满足上面的几个点。因此，大多数已有的 web 自动化框架都不能在项目中适用。因此，基于这种现实，淘宝自动化组根据淘宝的项目实际研发的 AutoMan 框架就很好的解决了这个问题。

AutoMan 核心思想是将“元素查找”和“控件操作”分开，即通过 web 化的方式对页面控件的查找进行管理，在编写脚本时选择事前定义好的控件进行操作即可，因此该策略允许在项目的不同阶段分步进行“元素查找”和“控件操作”。下面给出一个项目自动化的测试流程图 1-1，方便用户理解：

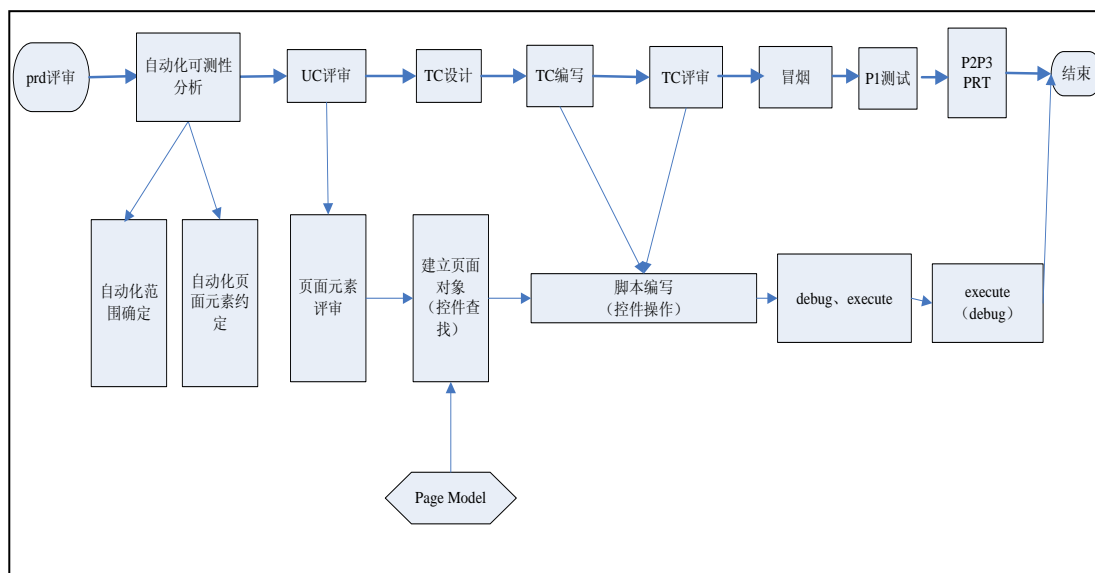


图 1-1 项目自动化的测试流程

通过上面的流程图，我们发现在项目的冒烟测试之前，可以根据开发提供的页面 DEMO 先将页面元素定义在 PageModel 上，实现初步的元素查找，然后将定义好的控件根据测试用例流程编写测试代码，及完成控件操作。当开发提供真正的页面之后，再完善元素查找，之后就可以将脚本执行起来。由此，1. 当遇到页面设计变化频繁时，我们只需要修改

PageModel 上定义的控件类型或者查找方式即可，通常不需要修改具体的脚本流程。2.由于是元素查找和控件操作是分步进行，并且在开发提供正式页面之前就进行，可以将整个测试时间大大提前，不需等到开发提供测试页面再进行脚本编写。因此很好的适应了项目周期短的问题。3.项目中一般会进行几轮测试，如果能将冒烟阶段或者第一轮测试阶段完成的脚本在后面几轮测试甚至是后面的回归测试中执行起来，那是自动化测试的价值体现。

综上所述，AutoMan 框架让项目自动化测试由不现实变成现实。

1.3 企业级的应用

企业级应用最大的特点是共享，将有限的资源共享给更多的用户，降低维护成本是页面自动化企业级应用的一大特点。AutoMan 框架很好的体现了共享这一点。主要体现在两方面的共享：1.PageModel 的共享，及页面元素查找的共享；2.脚本公用方法，公用流程的共享。

上文已经介绍 AutoMan 的核心是将元素查找和元素操作分离，即将相关的页面控件定义在 web 化的 PageModel 上。其目的一方面是为了降低脚本的元素查找维护，另一方面是实现页面元素的共享。举例子，淘宝的业务非常负责，流程繁多，但是无论卖家操作、买家操作还是交易管理都会先进行登陆操作。因此登陆是很多测试用例的前置条件。试想如果每条产品线都需要去对登陆页面进行查找操作显然是非常浪费的。正式由于 web 化的 PageModel 解决了这一问题，我们的方法是，只要有一条产品线对该页面进行建模，并创建好登陆共用方法，其他产品线都直接调用该定义好的页面或者方法即可，一旦页面发生变化，也只要由一条产品线同学进行维护，其他同学只管使用。类似这样的页面或者这样的共用方法都可以采用该操作，让不同产品线之间或者不同测试之间进行共享。因此通过共享大大降低了整个脚本的维护成本，实现企业级的应用。

1.4 AutoMan 的两大组件

AutoMan 通过将查找元素和控件操作分离的策略降低了脚本的维护成本、页面共享。因此作为 web 化管理对象库控件及查找配置的 PageModel 是 AutoMan 的一个重要组件。此外为了方便解决脚本调试，AutoMan 还提供了一个轻巧、便捷的调试小工具：AutoMan console。该工具支持用户脱离 IDE，在 CMD 命令行下进行元素控件的查找校验与代码调试等功能，下文会有具体介绍。只是基于 AutoMan 框架具有的上手简单、脚本编写快捷、调试方便才使得其框架在淘宝内部的日常回归和项目自动化测试中得到广泛推广和使用。

2 从这里开始

2.1 Automan-server 环境搭建

在试用的时候，可以连接 <http://automan.heroku.com/>直接体验。

标准方式下，用户使用自己搭建的环境。环境的搭建，参见 <http://www.github.com/automan> 下 automan-server 包的 readme 文件。

2.2 示例代码

参见 <http://www.github.com/automan> 下 automan-client 包的 example 文件夹。

3 PageModel

PageModel 是我们 AutoMan 的三大组件之一，用于将页面元素结构化的抽离出来，进行统一的页面对象库管理，它的最大优势是灵活应对页面变化。QTP 也有自己的对象库，但是它的对象库是在客户端的、并且是平铺式存放管理页面上控件。与 QTP 不同，PageModel 主要有三方面的优势：首先，允许用户根据使用的需要将页面控件有组织的、有结构的进行管理，方便对页面控件的组织；其次，允许用户在还不确定页面控件具体属性的时候，先在对象库创建控件结构，方便用户在项目中开发未提供具体页面时候就可以开始编写代码，提高效率；最后，采用 web 化的方式进行管理，实现对象库控件的共享，适合企业级的应用。


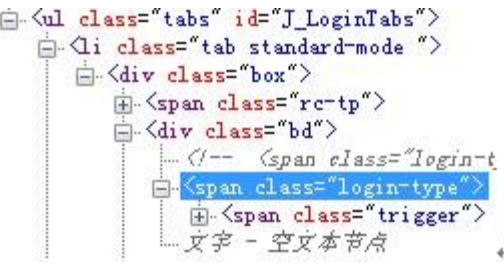

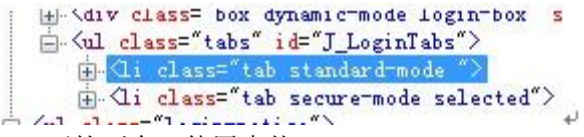
这里主要给大家介绍：用于查找和定位控件的 Selector；页面 PageModel 的两种组织方式 element 和 submodel；及对象控件的数组模式处理 collection。

3.1 Selector

selector 是用于查找、定位控件的查找器。支持 JQuery 语法。表 3-1 罗列了目前支持的所有 JQuery 方法。

表 3-1 Selector_JQuery 语法表

Selector	解释	示例
#id	用“#”表示目标元素的 Id	<INPUT id=on1 onclick=changeStuffStatus(5) value=5 type=radio> 查找 selector = “#on1”
.class	目标元素的 class, 用“.”表示 当 class 或 id 里面包含当属性里出现 #,&,.+*~:“!”^\$[]()=> / 或空格时, 应该加转义字符\\	<INPUT id=TPL_password class=login-text tabIndex=2 value="" type=password name=TPL_password> 查找 selector = “.login-text”（只是个例子，当然这里也可以用 id 来找的） <INPUT class="red login"> 查找 selector = “.red\\ login”
element	目标元素的 tag 名称，不区分大小写（一般不单独使用）	<IFRAME style="PADDING-BOTTOM: 0px; PADDING-LEFT: 0px; WIDTH: 628px; PADDING-RIGHT: 0px; HEIGHT: 350px; PADDING-TOP: 0px" id=_editor class=rtb marginHeight=1 frameBorder=1 marginWidth=1></IFRAME> 查找 selector="iframe" 或见下面的多重条件
多重条件	目标元素应当同时满足多个查询	<IFRAME style="PADDING-BOTTOM: 0px; PADDING-LEFT: 0px; WIDTH: 628px; PADDING-RIGHT: 0px; HEIGHT:

	条件	350px; PADDING-TOP: 0px" id=_editor class=rtb marginHeight=1 frameBorder=1 marginWidth=1></IFRAME> 查找 selector="iframe#_editor.rtb" 或 selector="iframe.rtb#_editor" 也可以少写点条件的, 能 唯一确定了就好了, 如 selector = "#_editor"
parent>child	目标元素是一个容易确定的元素的直接儿子	 找<a>查找 selector = "#logo>a"
ancestor descendant	目标元素是另一个容易确定的元素的后代(子孙), 使用空格	 找使用查找 selector="#J_LoginTabs .login-type"
层级查找	目标元素可以通过逐层查找得到	找使用查找 selector = "#register-form>p.register-btn img"
:eq()	目标元素不能唯一确定时, 可以指定是第几个。从 0 开始	 找第一个 li 下的第二个 a, 使用查找 selector = ".login-notice>li:eq(0)>a:eq(1)"
*	可代表任意 tag	不经常使用  找 ul 下的两个 li 使用查找 selector = "#J_LoginTabs>*", 这时 ul 下所有的儿子都会返回。当然, 也可以用 selector = "#J_LoginTabs>li"。多个元素的返回, 参见 Collection 填写说明
[name=value]	目标元素不能唯一确定时, 可用他的属性做过滤。当属性里出现 #, &, ., +, * ~ ! : " ! ^ \$ [] () = > / 或空格时, 应该加转义字符 \\	<INPUT name=names[] value=5> 查找 selector = "input[name=names\\[\\]]"

层级:		
+	匹配所有跟在 label 后面的 input 元素	<pre><form> <label>Name:</label> <input name="name" /> <fieldset> <label>Newsletter:</label> <input name="newsletter" /> </fieldset> </form> <input name="none" /></pre> <p>使用: "label+input"</p> <p>结果: [<input name="name" />, <input name="newsletter" />]</p>
~	找到所有与表单同辈的 input 元素	<pre><form> <label>Name:</label> <input name="name" /> <fieldset> <label>Newsletter:</label> <input name="newsletter" /> </fieldset> </form> <input name="none" /></pre> <p>使用: "form ~ input"</p> <p>结果: [<input name="none" />]</p>
基本:		
,	你可以指定任意多个选择器, 并将匹配到的元素合并到一个结果内。相当于“或”	<pre><div>div</div> <p class="myClass">p class="myClass"</p> span <p class="notMyClass">p class="notMyClass"</p></pre> <p>使用: "div,span,p.myClass"</p> <p>结果: [<div>div</div>, <p class="myClass">p class="myClass"</p>, span]</p>
属性:		
[name]	存在这个属性	如 input[id]返回所有存在 id 的 input 元素
[name^=value]	属性要以 value 开始	
[name!=value]	属性不为 value	
[name=value]	属性为 value, 如 input[name=q]	
[name\$=value]	属性以 value 结尾	
[name*=value]	属性包含 value	
过滤:		
:first	获取匹配的第一个元素, 相当	<pre> list item 1</pre>

	于:eq(0)	list item 2 list item 3 list item 4 list item 5 使用: "li:first" 结果: [list item 1]
:last	获取匹配的最后个元素	 list item 1 list item 2 list item 3 list item 4 list item 5 使用: "li:last" 结果: [list item 5]
:not	查找所有不包含 id 的 input 元素	<input name="apple" /> <input name="flower" id="checked" /> 使用: "input:not([id])" 结果: [<input name="apple" />]
:even	查找表格的索引值 0、2、4...	<table> <tr><td>Header 1</td></tr> <tr><td>Value 1</td></tr> <tr><td>Value 2</td></tr> </table> 使用: "tr:even" 结果: [<tr><td>Header 1</td></tr>, <tr><td>Value 2</td></tr>]
:odd	查找表格的索引值 1、3、5...	<table> <tr><td>Header 1</td></tr> <tr><td>Value 1</td></tr> <tr><td>Value 2</td></tr> </table> 使用: "tr:odd" 结果: [<tr><td>Value 1</td></tr>]
:gt	查找第二第三行, 即索引值是 1 和 2, 也就是比 0 大	<table> <tr><td>Header 1</td></tr> <tr><td>Value 1</td></tr> <tr><td>Value 2</td></tr> </table> 使用: "tr:gt(0)" 结果: [<tr><td>Value 1</td></tr>, <tr><td>Value 2</td></tr>]
:lt	查找第一第二行,	<table>

	即索引值是 0 和 1，也就是比 2 小	<pre><tr><td>Header 1</td></tr> <tr><td>Value 1</td></tr> <tr><td>Value 2</td></tr> </table></pre> 使用: "tr:lt(2)" 结果: [<tr><td>Header 1</td></tr>, <tr><td>Value 1</td></tr>]
内容:		
:contains	建设中.....还没有实现哦~ 查找所有包含 "John" 的 div 元素	<pre><div>John Resig</div> <div>George Martin</div> <div>Malcom John Sinclair</div></pre> 使用: "div:contains('John')" 结果: [<div>John Resig</div>, <div>Malcom John Sinclair</div>]
:empty	查找所有不包含子元素或者文本的空元素	<pre><table> <tr><td>Value 1</td><td></td></tr> <tr><td>Value 2</td><td></td></tr> </table></pre> 使用: "td:empty" 结果: [<td></td>, <td></td>]
:has	给所有包含 p 元素的 div 元素添加一个 text 类	<pre><div><p>Hello</p></div> <div>Hello again!</div></pre> 使用: "div:has(p)" 结果: [<div class="test"><p>Hello</p></div>]
:parent	查找所有含有子元素或者文本的 td 元素	<pre><table> <tr><td>Value 1</td><td></td></tr> <tr><td>Value 2</td><td></td></tr> </table></pre> 使用: "td:parent" 结果: [<td>Value 1</td>, <td>Value 2</td>]
子元素:		
:first-child	匹配第一个子元素 :first 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素	<pre> John Karl Brandon Glen Tane Ralph </pre> 使用: "ul li:first-child" 结果: [John, Glen]
:last-child	匹配最后一个子	<pre></pre>

	<p>元素</p> <p>:last 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素</p>	<pre>John Karl Brandon Glen Tane Ralph 使用: "ul li:last-child" 结果: [Brandon, Ralph]</pre>
:nth-child	<p>匹配其父元素下的第 N 个子或奇偶元素</p>	<pre> John Karl Brandon Glen Tane Ralph 使用: "ul li:nth-child(1)" 结果: [Karl, Tane]</pre>
:only-child	<p>如果某个元素是父元素中唯一的子元素，那将会被匹配</p> <p>如果父元素中含有其他元素，那将不会被匹配。</p>	<pre> John Karl Brandon Glen 使用: "ul li:only-child" 结果: [Glen]</pre>

3.2 Element & Submodel

Element 和 Submodel 是 pagemodel 的两种组织方式，前者是控件的最小操作单位，即下面不能再有子结点；后者是一个区块，下面可以有多个子结点。submodel 下面可以有多个 element 和 submodel。如图 3-2 所示：[MyCartPage](#) 这个页面下有一大的 submodel [my_cart_records](#)，下面可以包含多个子 submodel，每个子 submodel 下面会有各自的 element，如 shop_info 下面就有两个 element，分别是：chk_choose_shop 和 shop_mjs_info。显然用户可以通过这样的类似搭积木的方式，可以将一个平面 page 立体画起来，方便对控件的管理。至于将怎样的一个区域定义为一个 submodel 用户可以根据自己需要定义。

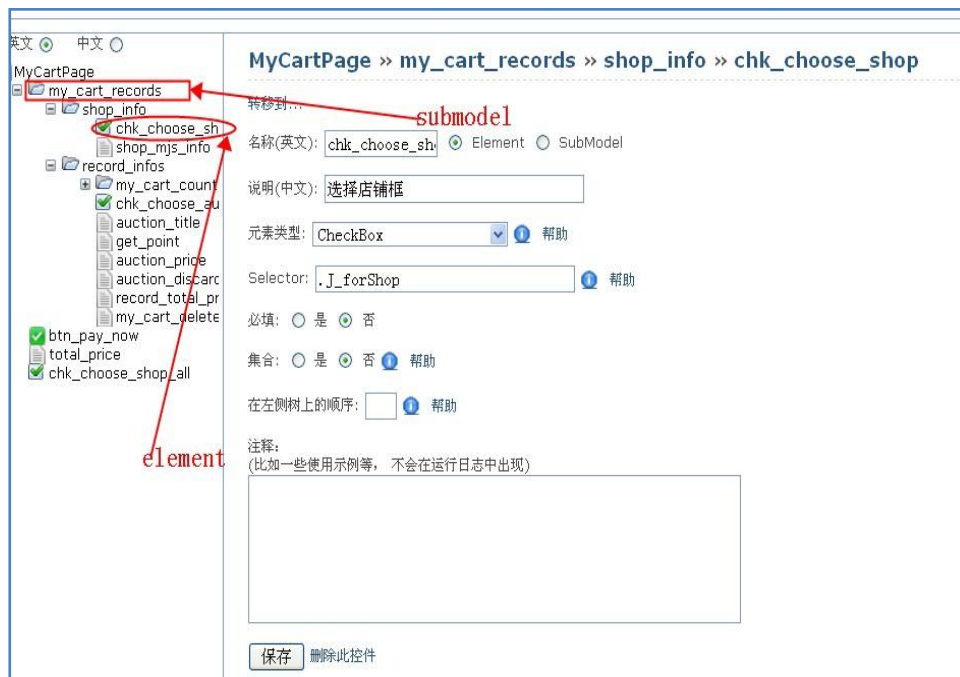


图 3-2 submodel 和 element 示意图

3.2.1 Element

Element 是页面控件的最小单位，在 PageModel 上配置 element 的时候会需要我们填写元素类型，因为不同的控件类型有不同的操作方法。例如：textfield 的控件会有 set 方法，而 button 类型的控件有 click 方法。另外如何判断一个页面控件是什么类型的，也是我们需要知道的，表 3-3 列出了新增 element 类型说明及具有的方法。

表 3-3 element 类型及方法

Element 类型	对应页面控件 tag	后台对应类型	方法	继承父类方法	使用场景
默认	Any (任何 tag)	AWatir::AElement	click, text, get, exist?		控件点击，取值等
Button	Button	AWatir::AButton	click	click, text, get, exist?	对 button 框的点击
CheckBox	Input(type=checkbox)	AWatir::ACheckBox	set,clear	click, text, get, exist?	对 CheckBox 框的选择和取消选择
Link	Li, A 等可以被点击的节点	AWatir::ALink	click	click, text, get, exist?	对 link 的点击
TextField	Input, text area, text	AWatir::ATextField	set	click, text, get, exist?	对普通输入框的输入操作
SelectList	Select	AWatir::ASelectList	set,selected_	click,	对下拉框的选择

t		st	value, options	text, get, exist?	操作
no_wait	会导致弹出框出现的节点, 如 button, input 等	AWatir::ANoWaitElement	click	text, get, exist?	当点击之后有系统弹出框的控件的点击操作
Radio	input(type = radio)	AWatir::ARadio	set,clear	click, text, get, exist?	对 Radio 框的选择和取消选择
rich_text	Body	AWatir::AInnerTextSetElement	set	click, text, get, exist?	对富文本框的输入操作

3.2.2 Submodel 与 element 的相对路径

我们需要查找一个 element 时需要填写 selector，而这个 selector 就是在一个范围内用什么方式能找到这个控件。所以当我们直接在这个 page 下定义了 element，那你填写的 selector 就是在这个页面内查找这个控件，当我们的 element 是定义在 submodel 下的，那我们的 selector 只要写在这个 submodel 下的能找到这个控件即可，这个就是相对路径查找。

举例说明：如图 3-4 所示，现在需要定位到控件“秋冬打底裤”

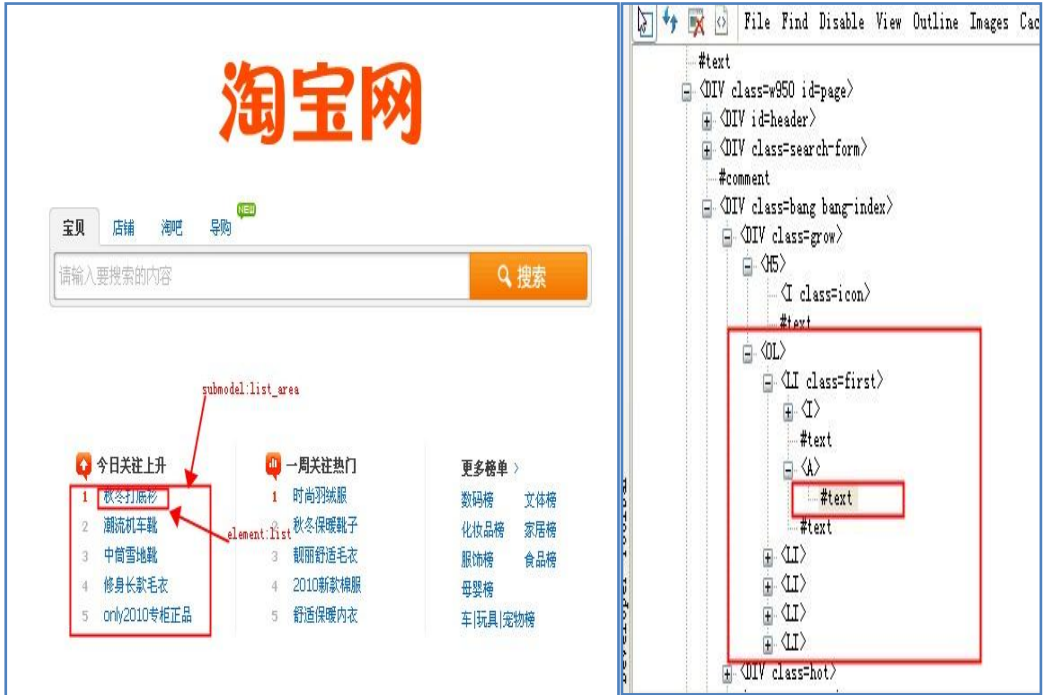


图 3-4 定义的 submodel 和 element

这里用两种方式查找“秋冬打底裤”

方法一：在页面上直接查找该控件 list，即该控件没有定义在任何一个 submodel 下。则该控件 list 的 selector 为 `div.grow>OL>li.first a`。

方法二：将 list 控件定义在 list_area 这个 submodel 下面。首先定义 list_area 的 selector 为 `div.grow>OL`。则 list 的 selector 为 `li.first a` 即可。

上面两个方法对比可以发现，如果一个控件是定义在一个 submodel 下的，那该控件的 selector 应该是相对 submodel 的一个相对路径，而不是一个完整的路径。

3.3 collection

collection 即集合，在我们填写对象库表单的时候会有一个是否是集合的选项就是对这个 collection 的操作。在我们平时操作的同一个页面上经常会遇到很多结构类似的布局，例如上图 3-4 所示的“今日关注上升”列表和“一周关注热门”列表，几个列表的结构都一致，都是有多条记录组成。如果按照之前 QTP 或者 Watir 方法进行控件定位的话我们需要对每个需要操作的控件进行定位。为了减少对这类类似布局的控件的重复定义，我们 pagemodel 这边引入了 collection（集合）这个概念，即将页面上一些相同结构的当作一个数组处理。并且数组可以是针对一个 submodel 下的所有控件的集合，例如图 3-4 上“今日关注上升”下面的所有记录当作一个数组。他也可以针对 submodel 的一个数组，例如图 3-4 的“今日关注上升”和“一周关注热门”当作一个数组。下面具体介绍下 element 的 collection 和 submodel 的 collection。

3.3.1 element 的 collection

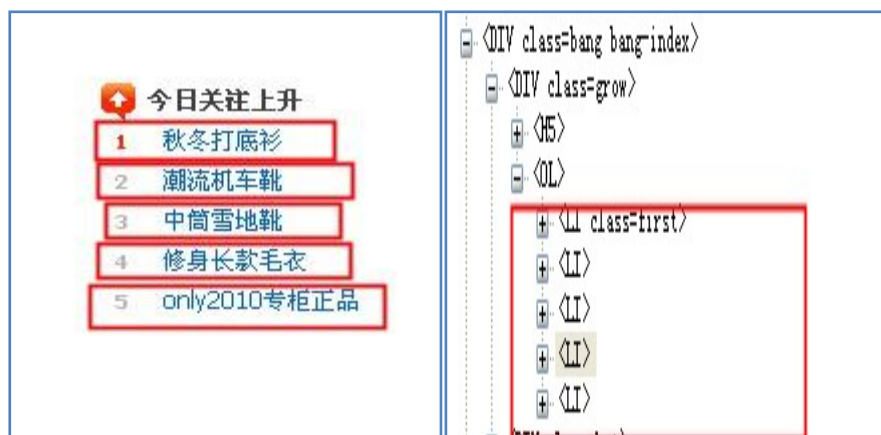


图 3-5 element 的 collection

上图是常见的一个列表形式的结构，从右图的结构上可以看出，上面的 5 条记录结构都一致，都是 OL 下面的 li 控件。假设现在需要在 5 个记录中选择一个，以前的方式是需要对所有的记录进行定义，而且在事前不知道要选那个记录的前提下，需要全部定义一次。那现在我们可以将这 5 个 li 认为是一个数组集合，只要定义一次，就可以找到数组中的任何一

个，即设置为 collection 的方式。对象库上的设置如图 3-6:

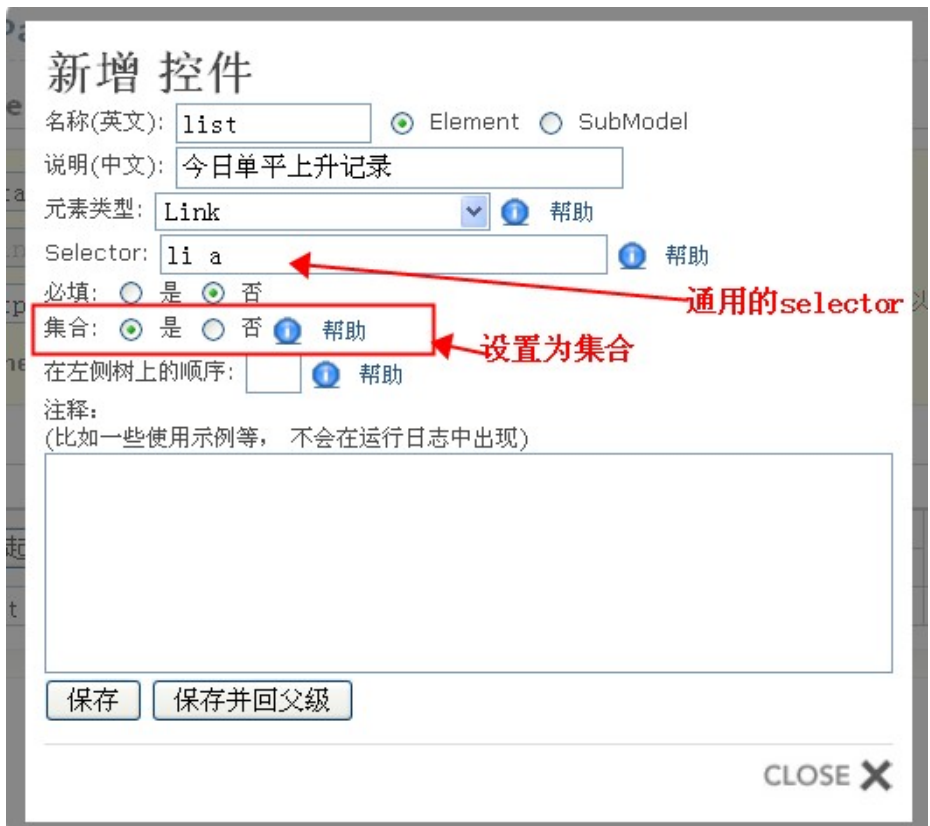


图 3-6 在平台上定义集合

假设现在需要选择一个记录，代码如下：

bpage.list_area.list[0].click	#选择第一个记录，即“秋冬打底衫”
bpage.rad_address[1].click	#选择第二个记录，即“潮流机车靴”

这里需要注意在填写 selector 时，一定要确保你的 selector 是能将数组中的控件都能找到的 selector。即是一个通用的 selector。同时在平台上设置的时候一定选择“集合”，如图 3-6 所示。

这里也支持用关键字的方式定位到具体的数据组中的某个记录。例如，上面的例子可能位置会变化，所以用序号的方式指定记录会出现不稳定。所以用文字来指定会更准确，代码如下：

bpage.list_area.list[“秋冬打底衫”].click	#选择第一个记录，即“秋冬打底衫”
bpage.rad_address[“潮流机车靴”].click	#选择第二个记录，即“潮流机车靴”

3.3.2 submodel 的 collection



图 3-7 submodel 的 collection

submodel 的 collection 和 element 类似，只不过前者是一个区块的集合，后者是一个控件的集合。图 3-7 可以看出，“今日关注上升”列表和“一周关注热门”结构一致，都是由一个 H5 和 OL 组成，并 OL 下面是具体的记录。所以根据 collection 的思想我们可以将“关注”信息定义为数组，数组包含了“今日关注上升”和“一周关注热门”，这样我们可以通过这个数组找到任何一条在这个表里面出现的订单信息。对象库上的设置如图 3-8:



图 3-8 对象库上 submodel 的 collection 定义

现在假设需要取某个关注区块的内容:

btext=bpage.list_area["今日关注"].text #btext="今日关注上升"下的所有内容

```
btext=bpage.list_area["一周关注"].text    #btext="一周关注热门"下的所有内容
```

由上面代码可以看出，对与 submodel 区块我们也可以根据结构是否一致定义为数组，方便数组取值，比之前需要一一罗列定位信息更灵活和方便。

现在将 submodel 的 collection 和 element 的 collection 集合起来定位一个控件。代码如下：

```
bpage.list_area["今日关注"].list["秋冬打底衫"].click #选"今日关注上升"下的“秋冬打底衫”  
bpage.list_area["一周关注"].list["舒适保暖内衣"].click #选"一周关注"下的“舒适保暖内衣”
```

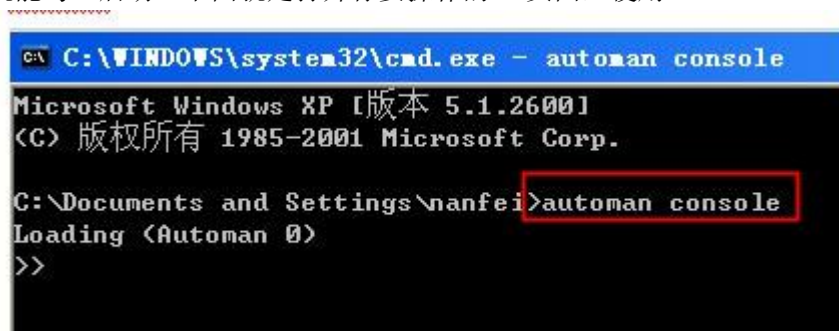
由上面的代码可以看出，通过设置 submodel 和 element 的 collection 方法，我们可以将图中的所有记录内容都定位到，类似二维数组位置定义具体的控件。这种方式比之前直接定位具体控件方便和灵活很多，也简单很多。因此，掌握好 submodel、element、collection 能帮助我们高效、精确的建立对象库，定位控件。

4 AutoMan console

Automan console 专门针对 AutoMan 框架设置的一款代码调试小工具，特点：快速，轻巧。你可以通过这个神奇的小工具，进行控件定位，检验控件 selector 书写的正确性，以及直接调式代码。

4.1 准备和安装

要使用 automan console，只需要安装 automan 即可，在 cmd 里运行：automan console 就能马上启动。下面就是打开你要操作的 IE 页面，使用 automan console 了。



4.2 主要功能

automan console 主要有元素查找、代码调试功能，另外还有一些辅助的小功能，例如将定义好的控件 mark 到页面上等。下面就具体分别介绍下。

4.2.1 元素查找

元素查找是 automan console 的最大应用，就是定位在页面上要操作的元素，校验该元素 selector 写法的正确性。对于元素的定位，工具提供三条命令：

1. find(url,selector, element_type)

(1) 函数介绍：

功能：返回页面上满足条件的所有元素
参数：url，目标 IE 的链接，支持正则匹配
selector，控件的查找方法
element_type，控件类型,默认为 AElement
返回值：array，所有满足条件的元素集合

(2) 函数使用举例：

我们需要是在淘宝首页 (<http://www.taobao.com>)，定位搜索栏上面的“宝贝”、“淘宝商城”、“店铺”、“拍卖”4 个连接。用 find(url,selector,element_type)查找 Selector: #J_TSearchTabs li，定位“id”为“J_TSearchTabs”元素下面所有为“li”的子孙。如图 4-1 所示，在 automan console 下使用命令 find(/taobao/, "#J_TSearchTabs li")后会显示所有满足条件的控件内容(即用红色框出的部分，共四个控件，分别对应页面上的：宝贝、淘宝商城、店铺、拍卖)。



图 4-1 find 方法演示

2. find_one(url,selector, element_type)

(1) 函数介绍:

功能: 返回页面上满足条件的第一个元素
 参数: url, 目标 IE 的链接, 支持正则匹配
 selector, 控件的查找方法
 element_type, 控件类型, 默认为 AElement
 返回值: string, 满足条件的第一个元素

(2) 函数使用举例:

```

from (irh):10
>> find_one(/taobao/,"#J_IsearchTabs li")
[成功]IE操作, attach(??-mix:taobao)) [标题]淘宝网 - 淘! 我喜欢
IE6的bug: body的parentElement是BASE
=>
[控件名] LI,
[class] tsearch-tabs-active,
[outerHtml]
<LI class=tsearch-tabs-active><A href="http://list.taobao.com/browse/cat-0.htm"
target=_self>宝贝</A><S class=rc-tp-l></S><S class=rc-tp-r></S>
>>

```

图 4-2 find_one 方法演示

与上面的例子不同,本方法只会返回满足 selector 条件的 4 个连接中的第一个“宝贝”,如图 4-2 所示。

3. show(element)

(1) 函数介绍:

功能: 在页面上红色边框高亮显示元素, 2 秒后边框消失
参数: element, AElement 控件, 支持 element 和 elements 以及 model、models。
返回值: 被框的元素个数

(2) 函数使用举例:



图 4-3 show 单个控件

show 单个控件: 上面的例子, 我们是定位显示了 1 个元素, 即在 automan console 下运行 show(ele)后马上会在页面上将该控件“宝贝”用红色框高亮出来。

4.2.2 代码调试

我们也可以将脚本中的代码语句放入 Automan console 中进行调试。步骤如下:

1. 我们要在 cmd 命令行, 进入本地 page 文件夹的父层目录, 加载所有的对象库 page, 然后再执行 automan console, (如果是你的对象库是下载到 C: \automan\base, 需

要 cd 到 C:\automan\base)

```
D:\automan_project1\auction>automan console
Loading <Automan 0>
>>
```

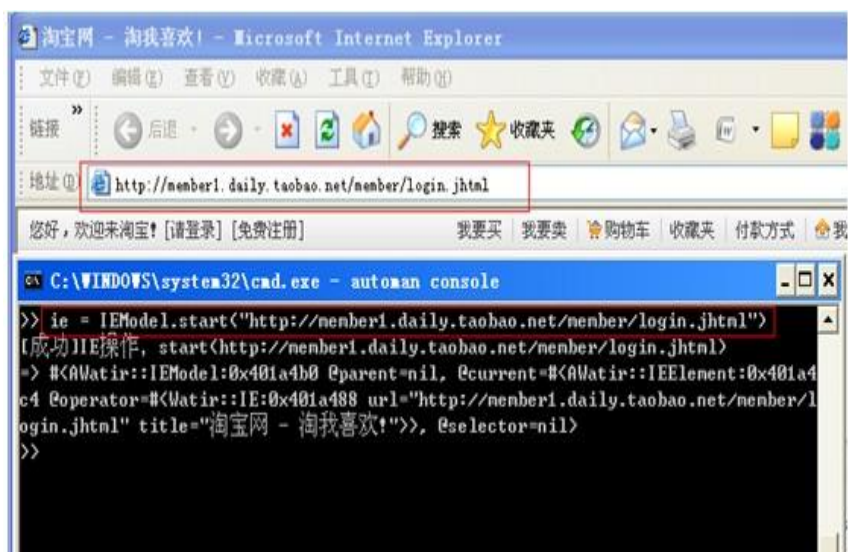
2. 进入 automan console 后，工具会加载该项目下的所有 page，直接输入 page 路径可以进行校验

```
D:\automan_project1\automan_project1>automan console
Loading <Automan 0>
<:AddDir=>"D:/automan_project1/automan_project1/page/Ttest">
<:AddDir=>"D:/automan_project1/automan_project1/page/Ttest/One">
>> IC::Login
=> IC::Login
>>
```



3. 开始调试

输入需要调试的脚本中的代码：



```
C:\WINDOWS\system32\cmd.exe - automan console
>> bpage = ie.cast(IC::Login)
[成功]页面模型操作, cast(IC::Login)
=> #<IC::Login:0x40134e4 @parent=#<AWatir::IEModel:0x401a4b0 @parent=nil, @current=#<AWatir::IEElement:0x401a4c4 @operator=#<Watir::IE:0x401a488 url="http://member1.daily.taobao.net/member/login.jhtml" title="淘宝网 - 淘我喜欢!">>, @selector=nil>, @current=
[控件名] HTML,
[outerHtml] <HTML xmlns="http://www.w3.org/1999/xhtml"><HEAD><TITLE>淘宝网 - 淘我喜欢!</TITLE>
<META http-equiv=Content-Type content="text/html; charset=gb2312">
<META http-equiv=X-UA-Compatible content=IE=?>
<META content="欢迎免费注册淘宝, 淘宝 - 亚洲最大的网上交易平台, 拥有近2亿商品, 提供安全便捷的购物体验, 提供先行赔付、假一赔三、7天无理由退货等售后服务, 同时还提供了各类网上赚钱的机会。" name=description>
<META content="会员注册 免费 淘宝 网上开店 网上购物 网上交易 网上商店 免费开店 网上买东西" name=keywords><LINK href="http://pics.taobao.com/favicon.ico" type=image/x-icon rel="shortcut icon"><LINK title=淘宝购物 href="http://assets.taobaocdn.com/plugins/opensource/provider.xml" type=application/opensourceprovider+xml rel=search><LINK href="http://assets.taobaocdn.com/tbsp/tbsp.css?t=20090602.css" rel=stylesheet>
<SCRIPT src="http://assets.taobaocdn.com/tbra/1.0/tbra-widgets.js?t=201003241751.js"></SCRIPT>
<LINK href="http://assets.taobaocdn.com/tbsp/global_nav.css?t=....., @selector=nil>
>>
```



通过命令行里面打出的日志，可以发现你的代码是否有问题。

4.2.3 其他功能

1. reload 方法

我们在上面介绍了，代码调式的时候需要进入本地的 page，然后启动 console 时会自动将平台上的 PageModel 下载到本地。然后才能在 console 下进行调试。那假如我们在调试的时候直接发错了平台上定义的控件有问题，直接在平台上修改了，那怎么让 console 能及时的拿到更新的 page 呢。这里只要轻松的在 console 下使用 load，即可。如图所示：

```
F=> #\Awatir::IEModel:0x3e9b468 @current=#\Awatir::IEElement:0x3e9fa7c @operator=#\Awatir::IE:0x3e9f16bc url= http://淘宝网 - 淘! 我喜欢">>, @parent=nil, @selector=nil>
irb(main):002:0> reload
UpdateFile=>c:/automan/base/page/Detail/AuctionDetail/BMallDetail.xml=>http://automan.taobao.net/api/pm_models/19
UpdateFile=>c:/automan/base/page/Taojianghu/Feeds/Feedslist.xml=>http://automan.taobao.net/api/pm_models/893.xml
UpdateFile=>c:/automan/base/page/WangWangClient/WWSystemSet/SignatureSet.xml=>http://automan.taobao.net/api/pm_models/2744.xml
AddFile=>c:/automan/base/page/Workflow/LargeAmount/DetailPage.xml=>http://automan.taobao.net/api/pm_models/2744.xml
=> nil
irb(main):003:0>
```

2. PageMarker.mark_model 功能

(1) 功能简介

前提：automan console 下获得 page 的实例，或者 SubModel 的实例。

用法：调用 PageMarker.mark_model p.search_area 或 PageMarker.mark_model p

作用：标明并框出该实例下面的所有页面模型，与 mark 的区别是这个方法可以只标出局部区域。

目前存在的问题：页面模型元素创建了很多以后，控件名会叠在一起；个别元素高亮不出来。

(2) 举例说明

在 automan console 下输入代码：

```
ie= IEModel.attach(/login/)
p= ie.cast(Mms::LoginPage)#这里也支持对 submodel 的 mark，即支持 p=page.submodel
PageMarker.mark_model(p)
```

则出现的效果如图，会将所有定义的控件在页面上显示出来，帮助用户编写代码。



如果觉得上面介绍的方法需要先 attach，然后在 cast 比较麻烦，而想通过直接 url 就可以对该页面进行 mark，也可以，这里提供一种的快捷方法，代码如下：

```
mark(/login/,Mms::Login)
```

效果和上面的一样，只是这里的参数不一致，第一个为页面的 url，第二个参数为该页面的命名空间。

3. HtmlModel 的实例方法 automan_methods

(1) 功能简介

前提：automan console 下获得 page 的实例，或者 SubModel 的实例。

用法：调用 p.search_area.automan_methods 或 p.automan_methods

作用：列出所有用户定义的方法

(2) 举例说明

在 automan console 下输入代码：

```
ie= IEModel.attach(/login/)
```

```
p= ie.cast(Mms::LoginPage) #这里也支持对 submodel 的 automan_methods，即支持
```

```
#s=page.submodel, s.automan_methods
```

```
p.automan_methods
```

则在 console 下会列出 P 下的所有定义的对象实例，如图所示：

```
irb(main):021:0> p.automan_methods
=> ["btn_login", "chk_safelogin", "lnk_forget_pwd", "txt_username", "txt_password"]
irb(main):022:0>
```