

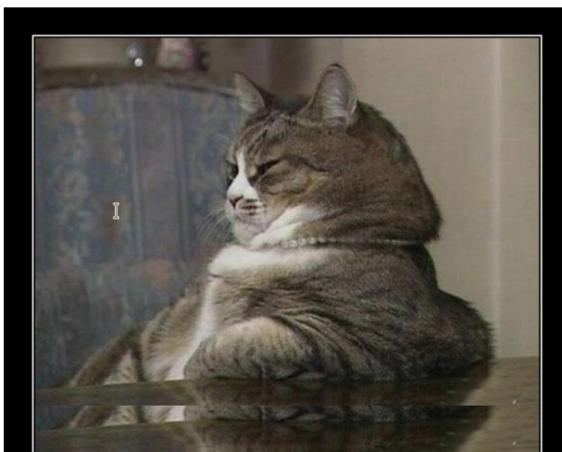


RadST 12 ноября в 20:57

XPath: что нужно делать, а что нет

Разработка веб-сайтов *, Тестирование IT-систем *, CSS *, HTML *, Тестирование веб-сервисов *

Привет, Хабр! В прошлый раз мы уже поднимали тему написания селекторов на XPath для автоматизации тестирования веб-сервисов. Сегодня мне хотелось бы поговорить о практиках работы с XPath. Этот пост будет о том, какие приемы хорошо работают, а каких вещей лучше избегать, если вы так же как и мы сделали выбор в пользу XPath. Всех заинтересованных прошу под кат, а если у вас есть свои уже проверенные временем ноу-хау, давайте делиться ими в комментариях.



ПРОДОЛЖАЙ. Я ЗАИНТРИГОВАН

Мы постоянно работаем с большими объемами тестов, и по мере роста количества заказов в команду приходят новые инженеры. Именно обучение стало поводом для размышлений над тем, что такое "хороший XPath", а что такое "плохой XPath".

На первый взгляд может показаться, что в этом вопросе нет ничего сложного: просто берете общепринятый стандарт для селекторов, сверху кладете документацию по XPath и отдаете все это новому сотруднику со словами: "Знакомься товарищ!". Но практика показала, что просто знаний синтаксиса недостаточно. И в работе встречаются как хорошие, так и плохие практики написания селекторов. Именно исходя из этого опыта и родился этот пост. А ниже вы найдете те принципы и практики, которые мы выработали сами для себя, набив несколько шишек, потратив часы лишнего времени на исправления и так далее.

XPath и как правильного его готовить

1) Удаляйте лишние пробелы в строке с помощью функции normalize-space().

Код `<div class="btn">Войти через Google</div>`Плохая практика `//[text()='Войти через Google']`Хорошая практика `//[normalize-space(text())='войти через Google']`

Почему: Последний селектор нивелирует ошибки в верстке, связанные с пробелами. Например, это спасет в подобной ситуации:

- `<div class="btn"> Войти через Google </div>`
- `<div class="btn"> Войти через Google </div>`

2) Не пишите селекторы по полному совпадению наименования классов.

Код `<div class="large red Menu_mainLink">Услуги</div>`Плохая практика `//div[@class='large red Menu_mainLink']`Хорошая практика `//div[contains(@class, 'Menu_mainLink')]`

Почему: Буквально завтра заказчик может поменять дизайн и вместо класса `red` использовать `blue`, или добавить/удалить какой-нибудь из классов. Если вы не хотите делать лишнюю работу лучше предусмотреть устойчивость селектора к подобным изменениям. А сделать это можно, отказавшись от явного указания на наименование класса.

3) Не используйте фильтры по номерам, если можно этого избежать.

Код:

```
<div class="large red Menu_mainLink">
  <a class="Menu_item_5wCCA">Услуги</a>
  <a class="Menu_item_5wCCB">Афиша</a>
```

Реклама

ЧИТАЮТ СЕЙЧАС

Яндекс.Практикум: самый подробный отзыв

26K 178 +178

Танк, которого нет в World of Tanks

19K 15 +15

Поговорим про собеседования: взгляд бэкендера

4.8K 18 +18

Опрос: миграция айтишников из Беларуси после событий 2020

13K 61 +61

«Аквариус» создал отечественный смартфон для военных за 1 млрд рублей

9.3K 29 +29

Настоящие изобретения, похожие на выдумку — и наоборот

Megatest

Настоящие изобретения, похожие на выдумку — и наоборот

Megatest

РАБОТА

Тестирующий программного обеспечения
149 вакансий

Все вакансии

```
<a class="Menu_item__5wCCC">Карта</a>  
</div>  
Пишем селектор к элементу <a>Карта</a>  
Плохая практика //div/a)[3]  
Хорошая практика //a[normalize-space(.)='Карта']
```

Почему: Количество пунктов в меню и их порядок могут поменяться по мере развития сайта.
Правильно составленный селектор будет гораздо устойчивее и окажется независим от количества пунктов в меню.

"Живой" пример "плохого" селектора:

```
//[@id='login_form']/div[2]/div/table/tbody/tr[2]/td[2]/input
```

4) Используйте один селектор для множества элементов.

Снова обратимся к коду из пункта №3. Если посмотреть внимательно, то один XPath селектор можно использовать для каждого пункта меню, достаточно обернуть его в функцию.

Плохая практика (писать почти одинаковые селекторы к каждому пункту):

```
service=driver.find_element_by_xpath(f"//normalize-space(.)='Услуги'")  
affiche=driver.find_element_by_xpath(f"//normalize-space(.)='Афиша'")  
map=driver.find_element_by_xpath(f"//normalize-space(.)='Карта'")
```

Хорошая практика:

```
def get_menu_item(text):  
    element=driver.find_element_by_xpath(f"//normalize-space(.)='{text}'")  
    return element  
  
service = get_menu_item('Услуги')  
affiche = get_menu_item('Афиша')  
map = get_menu_item('Карта')
```

Почему: Если не использовать общий селектор, то работы по написанию становится больше. А при грамотном оформлении функции получается удобно читаемый/редактируемый код, возрастает скорость его написания.

5) Используйте поиск по вложенному тексту().

Для тегов <h1><h2>, <a>, <button>, , <td> по возможности необходимо использовать поиск по вложенному тексту (), а не по text().

Код:

```
<button jsname="pzCKEC" class="EzVRq" aria-controls="dEjpnf" aria-haspopup="true">Настройки</button>  
  
Плохая практика //button[normalize-space(text())='Настройки']  
Хорошая практика: //button[normalize-space(.)='Настройки']
```

Почему: Разработчик в любой момент может обернуть текст внутри в какой-нибудь , и тогда поиск по text() не будет работать. Бывает, что фреймворки не могут кликнуть по вложенному тегу, иногда это приводит к ошибкам перехвата клика элементом уровнем выше. Если же искать по вложенному тексту, все эти проблемы перестают быть опасными.

6) Магия XPath для динамического контента

Эта ситуация еще интереснее. Предположим, перед вами стоит задача, определить наличие текста на странице. Но текст при этом — динамический. То есть элемент загружается сразу, а текст в нем немного позже.

Плохая практика:

```
time.sleep(5) # Принудительная пауза 5 секунд  
//div[contains(@class, 'card')][normalize-space(.)='Оглавление']
```

Хорошая практика: использование методов ожидания элемента в дереве DOM по "правильному" селектору, например вот такому:

```
//div[contains(@class, 'card')][not(normalize-space(.)=='')]
```

Система будет ждать, пока селектор не станет актуальным, т.е. в нашем случае, пока не появится текст.

Еще один пример **правильного** селектора для работы с динамическим текстом элемента. Здесь используем "|" (или):

```
//[normalize-space(text())='Войти'] | //normalize-space(text())='Авторизоваться']
```

Почему: Использование обычных пауз в коде не решает проблемы, так как сводится к лотерее - успеет или не успеет загрузиться. Методы ожидания элемента делают работу селектора на 100% независимой от склонности браузера подождать.

7) Используйте отношения элементов DOM

Мы уже говорили выше о возможности XPath двигаться по дереву DOM вверх и вниз. Но это не единственный пример хорошей практики использования отношений элементов.

Представьте, что у вас страница с множеством одинаковых блоков, в которых может содержаться множество одинаковых элементов. XPath позволяет определить конкретный блок по какому-либо признаку и далее осуществлять поиск элементов только в нем.

Код:

```
<form-field><input id="123">...</input><a>Скачать</a>
</form-field>

<form-field><input id="abc">...</input><a>Скачать</a>
</form-field>

<form-field><input id="xyz">...</input><a>Скачать</a>
</form-field>
```

Плохая практика:

```
//input[@id='123']/../../../../form-field//*[@normalize-space(text())='какой-то
текст']
```

Хорошая практика:

```
//form-field//input[@id='abc']//*[@normalize-space(text())='Скачать']
```

Здесь мы ищем текст только в определенном блоке `//form-field`, у которого есть `input` поле с `id=abc`.

Заключение

Подводя небольшой итог, хочется сказать, что XPath — мощный и эффективный инструмент, и после небольшого знакомства с документацией вы сможете с ним работать, так же, как и мы. Однако с осторожностью относитесь к всевозможным плагинам и стандартным средствам браузера при составлении селекторов. Не стоит использовать в селекторах фильтры с указанием номера элемента, стилистические теги (например, `<D>`, `<I>`, `` и т.п.) и так далее.

При этом от работы с XPath будет больше эффекта, если пользоваться встроенными функциями XPath и логическими операторами. Тестирование становится легче автоматизировать, если вы будете стремиться к короткому и однозначному селектору, использовать отношения элементов DOM, а также создавать универсальные селекторы и стандартизировать подход к составлению селекторов. Все это создает возможности для оперативного обновления селекторов и ускоряет подготовку тестов.

Использование всех этих рекомендаций позволяет нам качественно и быстро писать селекторы для реализации задач заказчика, достаточно оперативно обучать новый персонал, вносить изменения и масштабировать проекты. А если у вас уже есть свой опыт работы с селекторами XPath, я буду очень рад вашим примерам, лайф-хакам, комментариям, дополнениям и даже критике наших методов.

Теги: XPATH, CSS, HTML, тестирование, селекторы

Хабы: Разработка веб-сайтов, Тестирование IT-систем, CSS, HTML, Тестирование веб-сервисов

Редакторский дайджест
Присыпаем лучшие статьи раз в месяц

Электропочта

Sergey Radchenko @RadST
Руководитель отдела внедрения систем мониторинга
Сайт

Комментарии 12

monane 12.11.2021 в 21:54
Извините, а `//*[string-length(element)]?` Хотя о красоте это не совсем). Иногда при парсинге элементы динамические помогают и так если на стороне сервера сильно против).

quarkster 12.11.2021 в 23:33
Чтобы поменьше использовать xpath мы пришли к соглашению с разработчиками по использованию особой спецификации для атрибутов элементов в DOM <https://ouia.readthedocs.io> Идея в том, чтобы иметь предсказуемый селектор для любых компонентов в UI. Например, есть кнопка, которая в Html выглядит так:

```
<button class="someClass">Some Text</button>
```

В соответствии с нашей спецификацией кнопка должна иметь следующий вид:

```
<button data-ouia-component-type="Button" data-ouia-component-id="someId" class="someClass">
```

В итоге можно прийти к универсальному селектору:

```
.//*[@data-ouia-component-type=<имя компонента> and @data-ouia-component-id=<id компонента>]
```

Конечно, полностью уйти от произвольных селекторов не получится, но следуя соглашению, можно значительно упростить сопровождение тестов и повысить их стабильность.

♦ +4 Ответить

• o  **Foduch** 13.11.2021 в 02:06

У меня тоже был подобный опыт, я был в команде тестирования. Но в итоге пришлось от подобного отказаться: часто при рефакторинге или переворотке страницы разработчики могли полностью удалить блоки кода и написать заново, как итог атрибуты пропадали и тесты ссыпались.

Подход работает, но для этого нужны формальные договоренности писать дополнительные атрибуты и делать это всегда.

♦ +2 Ответить

• • o  **noodles** 13.11.2021 в 16:17

как итог атрибуты пропадали и тесты ссыпались

Так это ж значит что тест отработал себя, разве нет?

Что-то значительно поменялось - тест не прошёл. Значит нужно писать новый тест под новые требования.

♦ +2 Ответить

• • • o  **Foduch** 13.11.2021 в 16:35

То что он упал - хорошо. То из-за чего упал - плохо

Пример: на форме есть кнопка с текстом "Отправить" с атрибутом data-test="submit". После рефакторинга осталась такая же кнопка "Отправить", но атрибут пропал. В итоге тест упал, хотя кнопка на месте и форма работает. Мы могли искать, например, по тексту, тогда бы тест мог бы пройти успешно.

Поэтому дополнительные атрибуты только для тестов не всегда стабильно работают и нужно понимать риски их изменения или исчезновения

♦ +1 Ответить

• • • • o  **tundrawolf_kiba** 16.11.2021 в 19:56

По опыту — тесты из-за проблем с дополнительными атрибутами падают гораздо реже, чем из-за просто поменявшейся верстки.

♦ 0 Ответить

• • o  **quarkster** 20.11.2021 в 00:10

Мы пошли дальше и встроили необходимые нам атрибуты в нашу библиотеку компонентов <https://www.patternfly.org/v4/developer-resources/open-ui-automation>

♦ +1 Ответить

o  **klovov** 13.11.2021 в 01:43

А идея была хороша: сконструировать декларативный язык для произвольной выборки данных из иерархических древовидных структур. Вот как SQL для таблиц, а это будет такое же, но для деревьев. И, вообще-то, это получилось, но не прижилось. Наверное, для мэнинстрима это слишком сложно оказалось, как и Haskell.

♦ +4 Ответить

• o  **monane** 13.11.2021 в 14:51

Наверное не в сложности дело кмк, там нет ничего сложного. Причина банальна xpath без костылей невозможно использовать в браузере. Консоли dev tools и все или свой софт. И остается он только тем кто знает как его прикрутить к парсеру или разрабам.

♦ 0 Ответить

o  **kolob_vic** 16.11.2021 в 21:46

"Невилуйте проблемы в вёрстке и коде, подстраивайтесь под костыльный проект. Ну..лишь бы тесты работали"

♦ 0 Ответить

o  **leorush** 17.11.2021 в 12:33

"." - это не "вложенный текст". Это node. То есть весь элемент, включая теги.

И если как вы говорите "завтра заказчик может поменять дизайн", то почему не может появиться атрибут совпадающий с вашим текстом?

```
<button title="Настройки кое-чего другого">Остальные настройки</button>
```

♦ 0 Ответить

o  **mashy23** 19.11.2021 в 22:30