**Поиск элементов с помощью CSS-селекторов**

Ниже приведены части элементов HTML-страницы, по которым можно найти элемент:

* tag
* значение атрибута
* name
* class
* id

**Поиск по tag – Пишется просто сам тэг**

Чтобы найти элемент по тегу, просто напишите название тега в поисковой строке, как мы делали это при поиске по id (только без знака **#**), например, **h1**. Поиск по **h1** найдёт для нас элемент с названием страницы. Поиск по тегам не очень удобен, т.к. разработчики используют небольшое количество тегов для разметки страниц, и скорее всего, одному тегу будет соответствовать множество элементов.

**Поиск по значению атрибута – Пишется через [attribute=”value”]**

Можно найти элемент, указав название атрибута и его значение. Например, можно переписать поиск по id в следующем виде **[id="bullet"]**вместо **#bullet**.

Лучше использовать вариант с квадратными скобками при поиске значения атрибута для тех атрибутов, у которых нет собственных коротких команд поиска. Например, давайте найдем элемент h1 по значению его атрибута value: **[value="Cat memes"]**.

Лучше, что бы разработчики для каждого элемента создали **data** атрибут, который не изменяется, что бы в последующем в тестах можно было к ним обращаться

**Поиск по name**

Этот вариант поиска является разновидностью поиска по значению атрибута и записывается так же: **[name="bullet-cat"]**. Мы выделяем этот вариант потому что он довольно часто используется, а также выделяется как отдельный вид поиска элементов в Selenium WebDriver.

**Поиск по class – Пишется через .**

Поиск по классу можно записать в виде **[class="jumbotron-heading"]**, так как class тоже является атрибутом элемента. Но раз уж классы используются практически в каждой странице при задании стилей страниц, то для них также имеется свой короткий вариант поиска: **.jumbotron-heading**. То есть мы пишем значение класса и предваряем его точкой.

Давайте рассмотрим важную разницу между двумя способами поиска по классу. Допустим, у элемента **article** задано больше одного класса, как на странице <http://suninjuly.github.io/cats.html>:

<article id="moto" class="lead text-muted" title="one-thing" name="moto">If there's one thing that the internet was made for, it's funny cat memes.</article>

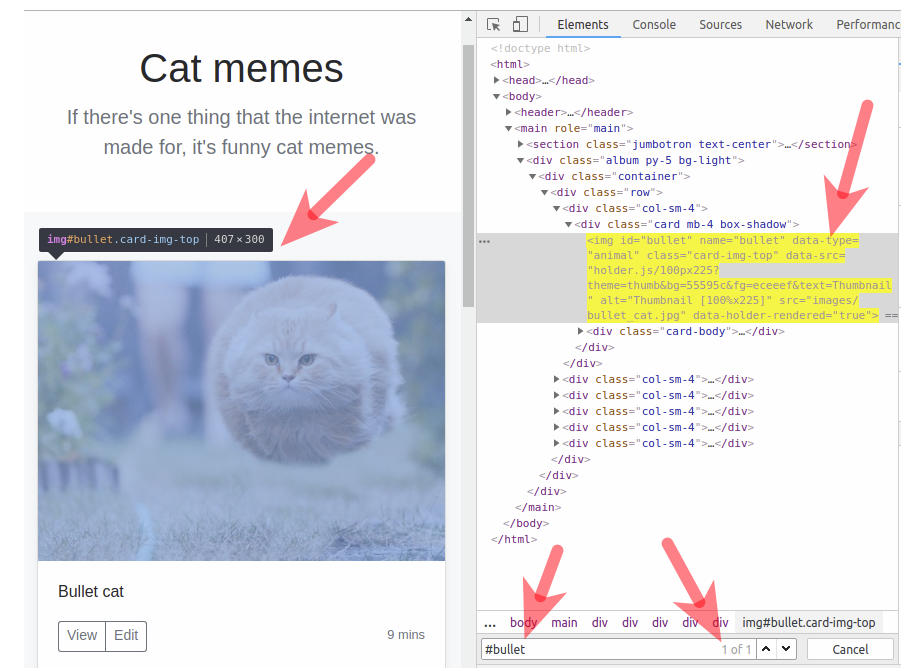
Вариант **[class="lead"]** не найдет нам этот элемент, так как он ищет по точному совпадению. Чтобы найти элемент, нам нужно будет написать **[class="lead text-muted"]**, порядок классов при этом важен. **[class="text-muted lead"]** — уже не найдет искомый элемент.

Вариант **.lead** при этом позволит найти данный элемент, так как он ищет простое вхождение класса в элемент. Для уточнения селектора можно задать также оба класса, для этого нужно добавить второй класс к строке поиска без пробела и предварить его точкой: **.lead.text-muted**. Порядок классов в отличие от первого способа здесь не важен — **.text-muted.lead** так же найдет нужный элемент. Рекомендуем пользоваться вторым способом поиска классов, так как он является более гибким.

Еще одно важное замечание. Поиск по классу чувствителен к регистру, то есть **.Lead** уже не найдет нужный элемент.

В консоли браузера вы также можете искать по простому совпадению текста в HTML, например, запрос **lead** подсветит текст **lead**. Однако, не стоит пользоваться таким поиском для выбора элементов, так как он слишком общий и не может использоваться в качестве селектора.

**Поиск по id – Пишется через #**



Какое везение! У элемента с нашей картинкой есть атрибут id="bullet", а значит, мы однозначно можем найти её с помощью селектора **#bullet**(знак **#** означает, что мы ищем по **id** со значением **bullet**).

*Можно проверить правильность подобранного селектора непосредственно в браузере в консоли разработчика. Откройте консоль разработчика и перейдите в ней на вкладку Elements. Затем нажмите ctrl+F и в открывшейся внизу поисковой строке введите селектор. Если селектор написан правильно, то вы увидите подсвеченный элемент на веб-странице, а также элемент будет подсвечен жёлтым цветом в html-коде. Еще в поисковой строке вы увидите количество найденных элементов. Желательно писать точные селекторы, которые позволяют найти ровно один элемент. В написании таких селекторов мы потренируемся в одной из следующих задач.*

*Еще один способ открыть консоль разработчика в браузере: нажать правой кнопкой мыши на любой элемент страницы и выбрать пункт меню "Посмотреть код" (англ. "Inspect") в контекстном меню. При этом на вкладке Elements сразу будет подсвечен кусок HTML-кода, описывающий данный элемент.*

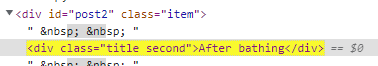
## Поиск элементов с помощью составных CSS-селекторов

Теперь предположим, что не можем найти элемент на странице, используя простой селектор, так как такой селектор находит сразу несколько элементов.

Вопрос: как нам найти селектор для подписи у второй картинки? Вот здесь нам поможет иерархическая структура страницы и возможность комбинировать CSS-селекторы. CSS-селекторы позволяют использовать одновременно любые селекторы, рассмотренные ранее, а также имеют некоторые дополнительные возможности для уточнения поиска.

**Использование потомков – все вложенные элементы**

#post2 .title



Здесь символ **#** означает, что надо искать элемент с id post2, пробел - что также нужно найти элемент-потомок, а **.**, что элемент-потомок должен иметь класс со значением title.

Элемент .title называется **потомком** (англ. **descendant**) элемента #post2. Потомок может находиться на любом уровне вложенности, все элементы с селектором .title также являются и потомками элемента #posts, хотя и расположены от него на два уровня ниже. #posts .title найдет все 3 элемента с классом title.







!Внимание. Символ пробела " " является значащим в CSS-селекторах. Это важный символ, который разделяет описание предка и потомка. Если бы мы записали селектор #post2.title без пробела, то в данном примере не было найдено ни одного элемента. Такая запись означала бы, что мы хотим найти элемент, который одновременно содержит id "post2" и класс "title". Таким образом #post2 .title и #post2.title — это разные селекторы**.**

**Использование дочерних элементов – все вложенные элементы на 1 уровень ниже**

Другой способ найти этот элемент:

#post2 > div.title

Здесь мы указали еще тег элемента divи уточнили, что нужно взять элемент с тегом и классом: div.title, который находится строго на один уровень иерархии ниже чем элемент #post2. Для этого используется символ >.

Элемент #post2 в этом случае называется **родителем** (англ. **parent**) для элементаdiv.title, а элемент div.title называется **дочерним элементом** (англ. **child**) для элемента #post2. Если символа > нет, то будет выполнен поиск всех элементов div.title на любом уровне ниже первого элемента.

!Внимание. В данном случае символы пробела вокруг символа ">" не несут важного значения в отличие от предыдущего примера, и могут быть опущены. Запись #post2>div.titleаналогична записи #post2 > div.title.

**Использование порядкового номера дочернего элемента**

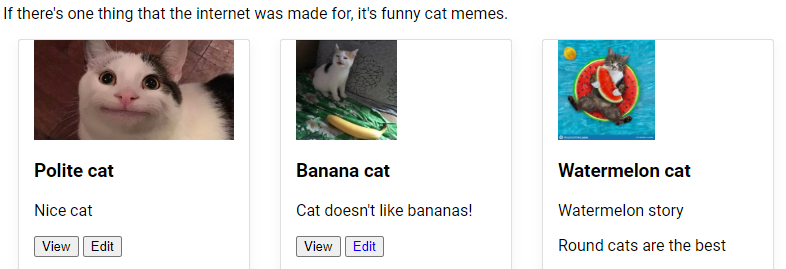
Еще один способ найти этот элемент:

#posts > .item:nth-child(2) > .title

Псевдо-класс :nth-child(2) — позволяет найти второй по порядку элемент среди дочерних элементов для #posts. Затем с помощью конструкции > .title мы указываем, что нам нужен элемент .title, родителем которого является найденный ранее элемент .item.

**Например**:

Нужно сделать кнопку синей

****

**.card-body:nth-child(2) .btn-group .btn.btn-sm:nth-child(2)**

1. **Выбираем второй элемент div, у которого класс card-body**
2. **Выбираем див элемент с классом btn-group**
3. **Выбираем второй элемент кнопки с классом btn btn-sm**

**Использование нескольких классов**

Также мы можем использовать сразу несколько классов элемента, чтобы его найти. Для этого классы записываются подряд через точку: .title.second

Мы рассмотрели базовые селекторы, которых будет достаточно для написания простых UI-тестов. Если вы захотите разобраться подробнее в css-селекторах, то мы рекомендуем вам посмотреть следующие статьи:

## Поиск элементов с помощью XPath

В работе с веб-страницами не всегда получается найти селектор, однозначно описывающий путь к нужному элементу. В такой ситуации для тестировщика лучшим решением проблемы будет пойти к фронтенд-разработчику проекта и договориться о специальном атрибуте, который будет использоваться в автотестах. Таким образом можно повысить тестируемость приложения. Увы, проекты бывают разные, и не всегда это возможно. И когда другого выхода больше нет, а автоматизировать как-то надо, можно обратиться к помощи языка запросов **XPath**.

### 1. XPath запрос всегда начинается с символа / или //

Символ / аналогичен символу > в CSS-селекторе, а символ // — пробелу. Их смысл:

* el1/el2 — выбирает элементы el2, являющиеся прямыми потомками el1;
* el1//el2 — выбирает элементы el2, являющиеся потомками el1 любой степени вложенности.

Разница состоит в том, что в XPath, когда мы начинаем запрос с символа /,  мы должны указать элемент, являющийся корнем нашего документа. Корнем всегда будет элемент с тегом <html>. Пример: /html/body/header

Мы можем начинать запрос и с символа //. Это будет означать, что мы хотим найти всех потомков корневого элемента без указания корневого элемента. В этом случае, для поиска того же хедера, мы можем выполнить запрос //header, так как других заголовков у нас нет.

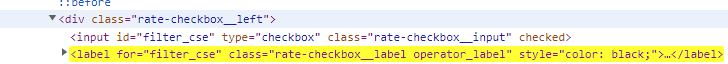
Важно! Такой поиск может быть неоднозначным. Например, запрос //div вернет вам все элементы с тегом <div>. Избегайте неоднозначных ситуаций, они плохо влияют на здоровье ваших автотестов.

### 2. Символ [ ] — это команда фильтрации

Если по запросу найдено несколько элементов, то будет произведена фильтрация по правилу, указанному в скобках.

Правил фильтрации очень много:

* по любому **атрибуту**, будь то id, class, title (или любой другой). Например, мы хотим найти картинку с летящим котом, для этого можно выполнить запрос //img[@id='bullet']
* по **порядковому номеру**. Допустим, мы хотим выбрать вторую по порядку карточку с котом. Для этого найдем элемент с классом "row" и возьмем его второго потомка: //div[@class="row"]/div[2]
* по **полному совпадению текста.** Да, XPath — это единственный способ найти элемент по внутреннему тексту. Если мы хотим найти блок текста с котом-Лениным, можно воспользоваться XPath селектором //p[text()="Lenin cat"]. Такой селектор вернет элемент, только если текст полностью совпадет. Здесь важно сказать, что не всегда поиск по тексту — это хорошая практика, особенно в случае мультиязычных сайтов.
* по **частичному совпадению** текста или атрибута. Для этого нужна функция contains. Запрос //p[contains(text(), "cat")] вернет нам все абзацы текста, которые содержат слово cat. Точно так же можно искать по частичному совпадению других атрибутов, это удобно, если у элемента несколько классов. Посмотрите на код навбара сайта с котами. Его можно найти селектором //div[contains(@class, "navbar")]
* в фильтрации еще можно использовать булевы операции (and, or, not) и некоторые простые арифметические выражения (но вообще не стоит, наверное). Допустим, мы хотим найти картинку обязательно с data-type "animal" и именем "bullet-cat", для этого подойдет запрос: //img[@name='bullet-cat' and @data-type='animal']
* //div[@class='rate-checkbox']//input**/following-sibling::**label – с помощью following-sibling::*element\_name* можно указывать что взять следующего соседа

**

Данная конструкция найдет label, только если предыдущий элемент был input

### 3. Символ \* — команда выбора всех элементов

* Например можем найти текст в заголовке запросом //div/\*[@class="jumbotron-heading"]. Это может быть удобно, когда мы не знаем точно тег элемента, который ищем.

### 4. Поиск по классу в XPath регистрозависим

Также как и в случае поиска по CSS-селектором будьте внимательными к регистру при поиске по классам:

**//div/\*[@class="Jumbotron-heading"]** не найдет элемент на нашей странице.

Что важно знать про XPath, чтобы пользоваться им безболезненно:

* Не используйте селекторы вида //div[1]/div[2]/div[3] без крайней нужды: по такому селектору невозможно с первого раза понять, что за элемент вы ищете. А когда структура страницы хоть немного изменится, то ваш селектор с большой вероятностью перестанет работать;
* Если есть возможность использовать CSS-селекторы: сlass, id или name — лучше использовать их вместо поиска по XPath;
* Можно искать по полному или частичному совпадению текста или любого атрибута;
* Можно использовать булевы операции и простую арифметику;
* Можно удобно перемещаться по структуре документа (переходить к потомкам и к родителям);
* Подойдет, когда у сайта всё плохо с атрибутами и нет возможности достучаться до разработчиков;
* Есть мнение, что поиск по XPath в среднем медленнее, чем по css. Но достоверно это неизвестно;
* Не стоит использовать разные расширения для браузеров по поиску XPath: они подбирают нечитабельные и переусложненные селекторы. Лучше потратить немного времени и разобраться в синтаксисе самостоятельно, тем более, что он не очень сложный.

**Как писать хорошие локаторы**

**1. Хороший локатор должен быть устойчив к изменениям.**

Это, пожалуй, самое главное. Если после небольшого изменения тестируемого приложения все тесты приходится переделывать – это просто катастрофа.

Есть изменения, которые изначально заложены в приложение и могут определяться его настройками – локализация, расположение блоков на странице (например, включить/отключить рекламные блоки), “шкурки” (skins) и прочее. Локаторы нужно сразу проектировать так, чтобы они работали с разными вариантами настроек.

Есть изменения, которые случатся в будущем. Тут заранее подготовиться на 100% нельзя, но можно хотя бы постараться предугадать, с чем эти изменения будут связаны. Например, если сейчас пока приложение одноязычное, но есть планы выхода на новые рынки – вероятно, добавятся разные локализации, поэтому поиск элементов по тексту перестанет работать. Если дизайнер скажет, что кнопки нужно сделать не прямоугольными, а закруглёнными – поменяются стили и вёрстка кнопок, поэтому полагаться при поиске на конкретный стиль рискованно, лучше выбрать какие-то более стабильные признаки.

Конечно, при массированном изменении тестируемого приложения противостоять изменениям в тестах не удастся, но это вполне ожидаемо. Но незначительные изменения не должны приводить к разрушению тестов.

**2. Хороший локатор должен быть понятным.**

Когда вы разрабатываете тесты, вы помните структуру страницы и можете легко представить, что найдётся по тому или иному локатору. Но через несколько месяцев память будет уже не так свежа, и глядя на запутанный локатор будет уже не так легко догадаться, что он ищет или хотя бы что он должен искать.

Конечно, можно добавить комментарии и пояснения. Но лучше всё таки стремиться к выразительности и понятности самих локаторов. Они должны быть простыми и осмысленными. Тогда читать и понимать старые тесты будет легко и приятно.

**3. Хороший локатор должен быть.**

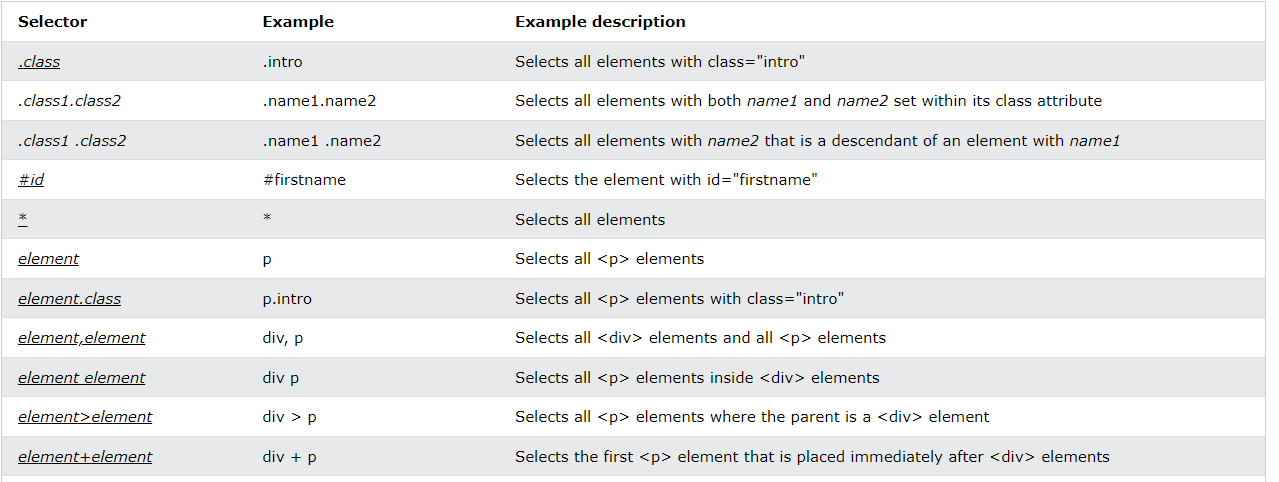
Увы, иногда хороший локатор вообще не удаётся построить. Вёрстка страницы плывёт, свойства элементов постоянно меняются. Если пытаться написать простой локатор – он оказывается неустойчивым, уже на следующий он не работает. В попытках повысить устойчивость повышаем сложность локаторов – но уже не через месяц, а через неделю становится непонятно, что они ищут, потому что прочитать локатор длиной в две строчки вообще невозможно.

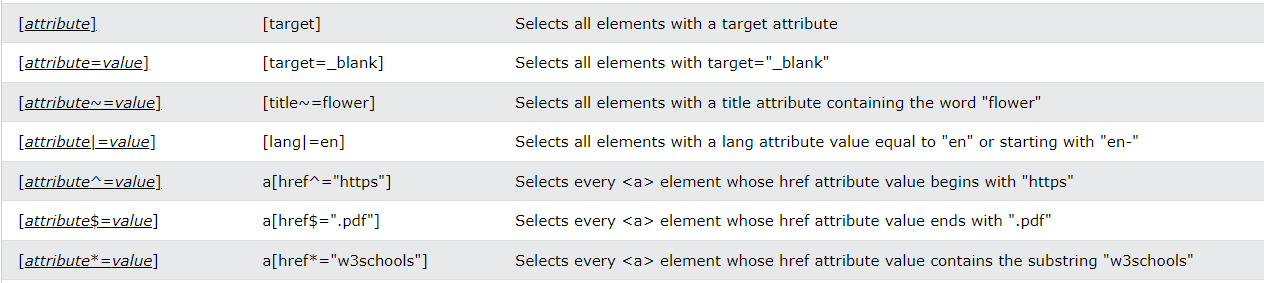
И тогда у тестировщика опускаются руки, он говорит “приложение невозможно автоматизировать” и уходит.

Это неправильно. Нужно бороться за свои права.

Идите к разработчикам и договаривайтесь о повышении тестопригодности приложения. Вообще-то они хорошие люди, просто они думали о своих проблемах и не думали о ваших. Если вы не расскажете им о том, что вам нужно от приложения – как они узнают?

**Селекторы CSS**

****

****

****