6. Внутреннее устройство поисковых методов

Несмотря на схожесть в синтаксисе, поисковые методы `get*` и `querySelector*` внутри устроены очень по-разному.
document.getElementById(id)
Браузер поддерживает у себя внутреннее соответствие `id -> элемент`. Поэтому нужный элемент возвращается сразу. Это очень быстро.
elem.querySelector(query), elem.querySelectorAll(query)
Чтобы найти элементы, удовлетворяющие поисковому запросу, браузер не использует никаких сложных структур данных.
Он просто перебирает все подэлементы внутри элемента `elem`(или по всему документу, если вызов в контексте документа) и проверяет каждый элемент на соответствие запросу `query`.
Вызов `querySelector` прекращает перебор после первого же найденного элемента, а `querySelectorAll` собирает найденные элементы в «псевдомассив»: внутреннюю структуру данных, по сути аналогичную массиву JavaScript.
Этот перебор происходит очень быстро, так как осуществляется непосредственно движком браузера, а не JavaScript-кодом.
Оптимизации:

- В случае поиска по ID: `elem.querySelector('#id')`, большинство браузеров оптимизируют поиск, используя вызов `getElementById`.
- Последние результаты поиска сохраняются в кеше. Но это до тех пор, пока документ как-нибудь не изменится.

elem.getElementsBy*(...)
Результаты поиска `getElementsBy*` – живые! При изменении документа – изменяется и результат запроса.

Способ Firefox
Перебрать подэлементы `document.body` в порядке их появления в поддереве. Запоминать *все найденные элементы* во внутренней структуре данных, чтобы при повторном обращении обойтись без поиска.
Разбор действий браузера при выполнении кода выше:

1. Браузер создаёт пустую «живую коллекцию» `elems`. Пока ничего не ищет.
2. Перебирает элементы, пока не найдёт первый `div`. Запоминает его и возвращает.
3. Перебирает элементы дальше, пока не найдёт элемент с индексом `995`. Запоминает все найденные.
4. Возвращает ранее запомненный элемент с индексом `500`, без дополнительного поиска!
5. Продолжает обход поддерева с элемента, на котором остановился (`995`) и до конца. Запоминает найденные элементы и возвращает их количество.

Способ WebKit
Перебирать подэлементы `document.body`. Запоминать только один, *последний найденный*, элемент, а также, по окончании перебора – длину коллекции.
Здесь кеширование используется меньше.
Разбор действий браузера по строкам:

1. Браузер создаёт пустую «живую коллекцию» `elems`. Пока ничего не ищет.
2. Перебирает элементы, пока не найдёт первый `div`. Запоминает его и возвращает.
3. Перебирает элементы дальше, пока не найдёт элемент с индексом `995`. Запоминает его и возвращает.
4. Браузер запоминает только последний найденный, поэтому не помнит об элементе `500`. Нужно найти его перебором поддерева. Этот перебор можно начать либо с начала – вперёд по поддереву, 500-й по счету) либо с элемента `995` – назад по поддереву, 495-й по счету. Так как назад разница в индексах меньше, то браузер выбирает второй путь и идёт от 995-го назад 495 раз. Запоминает теперь уже 500-й элемент и возвращает его.
5. Продолжает обход поддерева с 500-го (не 995-го!) элемента и до конца. Запоминает число найденных элементов и возвращает его.

Основное различие – в том, что Firefox запоминает все найденные, а Webkit – только последний. Таким образом, «метод Firefox» требует больше памяти, но гораздо эффективнее при повторном доступе к предыдущим элементам.
А «метод Webkit» ест меньше памяти и при этом работает не хуже в самом важном и частом случае – последовательном переборе коллекции, без возврата к ранее выбранным.
Запомненные элементы сбрасываются при изменениях DOM.
Документ может меняться. При этом, если изменение может повлиять на результаты поиска, то запомненные элементы необходимо сбросить. Например, добавление нового узла `div` сбросит запомненные элементы коллекции `elem.getElementsByTagName('div')`.
Сбрасывание запомненных элементов при изменении документа выполняется интеллектуально.

1. Во-первых, при добавлении элемента будут сброшены только те коллекции, которые могли быть затронуты обновлением. Например, если в документе есть два независимых раздела `<section>`, и поисковая коллекция привязана к первому из них, то при добавлении во второй – она сброшена не будет.
   Если точнее – будут сброшены все коллекции, привязанные к элементам вверх по иерархии от непосредственного родителя нового `div` и выше, то есть такие, которые потенциально могли измениться. И только они.
2. Во-вторых, если добавлен только `div`, то не будут сброшены запомненные элементы для поиска по другим тегам, например `elem.getElementsByTagName('a')`.
3. …И, конечно же, не любые изменения DOM приводят к сбросу кешей, а только те, которые могут повлиять на коллекцию. Если где-то добавлен новый атрибут элементу – с кешем для `getElementsByTagName` ничего не произойдёт, так как атрибут никак не может повлиять на результат поиска по тегу.