# Основные понятия и синтаксис

Адресная строка браузера это, пожалуй, наиболее чокнутая часть пользовательского интерфейса в мире. Адреса сайтов есть на рекламных щитах, на поездах и даже на уличных граффити. В сочетании с кнопкой «Назад» — наиболее важной кнопкой в браузере — у вас есть мощный способ двигаться вперед и назад через огромное множество взаимосвязанных ресурсов называемых вебом.

API истории HTML5 представляет собой стандартизированный способ манипулировать историей браузера через скрипт. Часть этого API — навигация по истории — была доступна в предыдущих версиях HTML. Новые части в HTML5 включают способ добавления записей в историю браузера, чтобы заметно изменить URL в адресной строке браузера (без переключения обновления страницы) и события, которые запускаются, когда эти записи удаляются из стека пользователя нажатием кнопки браузера «Назад». Это означает, что URL в адресной строке браузера может продолжать выполнять свою работу как уникальный идентификатор для текущего ресурса, даже в приложениях нагруженными скриптами, которые не всегда выполняют полное обновление страницы.

# Почему

Почему бы вам вручную не изменять адресную строку браузера? В конце концов, простая ссылка может перейти на новый URL, этот способ работал в течение 20 лет. И он будет продолжать работать таким образом. Этот API не пытается подорвать веб. Как раз наоборот. В последние годы веб-разработчики нашли новые и увлекательные способы подрыва веба без какой-либо помощи со стороны новых стандартов. API истории HTML5 на самом деле предназначен для того, чтобы адреса продолжали быть полезными в веб-приложениях нагруженными скриптами.

Возвращаясь к изначальным принципам, что теперь делает URL? Он определяет уникальный ресурс. Вы можете указать ссылку напрямую, добавить ее в закладки, поисковые системы ее индексируют, вы можете скопировать и вставить ее, отправить по почте кому-то еще, кто может нажать ссылку и в конечном итоге увидит тот же ресурс, что и вы первоначально. Все эти прекрасные качества URL сохраняются.

Поэтому мы хотим, чтобы уникальные ресурсы имели уникальный URL. Но в то же время браузеры всегда имели фундаментальное ограничение: если вы измените URL, даже через скрипт, это включит запрос к удаленному веб-серверу и приведет к обновлению страницы. Это требует времени и ресурсов, что, кажется, особенно расточительно, когда вы перемещаетесь на страницу, которая в значительной степени похожа на текущую. Все на новой странице будет загружено, включая те части, которые точно такие же, как на текущей странице. Нет способа сказать браузеру изменить URL и загрузить только половину страницы.

API истории HTML5 позволяет сделать это. Вместо запуска полного обновления страницы вы можете использовать скрипт, который, в сущности, скачивает половину страницы. Эта иллюзия довольно хитрая для воплощения и потребует некоторых усилий с вашей стороны.

Скажем, у вас есть две страницы, страница А и страница Б. Две страницы на 90% идентичны и только 10% содержимого страниц различается. Пользователь переходит на страницу А, затем пытается перейти к странице Б. Но вместо запуска полного обновления страницы, вы прерываете эту навигацию и совершаете следующие шаги вручную:

Загружаете 10% из страницы Б, которые отличаются от страницы А (возможно с помощью XMLHttpRequest). Это потребует некоторых серверных изменения в вашем веб-приложении. Вам нужно будет написать код, который возвращает только 10% от страницы Б, отличающихся от страницы А. Это может быть скрытый URL или параметр запроса, невидимый конечному пользователю.

Обмениваете измененное содержание (с использованием innerHtml или других методов DOM). Вам также может понадобиться сбросить любой обработчик событий для элемента внутри обменного содержания.

Обновляете строку браузера с адресом страницы Б, используя особый метод из API истории HTML5, что я вам покажу в данный момент.

В конце этой иллюзии (если выполнена правильно) браузер получает DOM идентичный странице Б, как если бы вы перешли страницу Б напрямую. Строка браузера будет содержать URL, который идентичен странице Б, как если бы вы перешли на страницу Б напрямую. Но в действительности вы не переходили на страницу Б и не делали полного обновления страницы. Это иллюзия. Но поскольку «компилированная» страница выглядит так же, как страница Б и имеет тот же URL, что у страницы Б, пользователь ни за что не заметит разницы (и не оценит ваш тяжелый труд по микроуправлению этого опыта).

# Как

API истории HTML5 это просто горстка методов объекта window.history плюс одно событие в объекте window. Вы можете использовать их, чтобы определить поддержку для API истории. Поддержка в настоящее время ограничивается самыми последними версиями некоторых браузеров, помещая эти методы прямо в лагерь «прогрессивного улучшения».

Поддержка истории

* IE 9.0
* Firefox 4.0+
* Safari 5.0+
* Chrome 8.0+
* Opera 11.10
* iPhone 4.2.1+
* Android -

History API опирается на один DOM интерфейс — объект History. Каждый таб имеет уникальный объект History, который находится в window.history. History имеет несколько методов, событий и свойств, которыми мы можем управлять из JavaScript. Каждая страница таба(Document object) представляет собой объект коллекции History. Каждый элемент истории состоит из URL и/или объекта состояния (state object), может иметь заголовок (title), Document object, данные форм, позиция скролла и другую информацию, связанную со страницей.

## Основные методы объекта History:

* window.history.length: Количество записей в текущей сессии истории
* window.history.state: Возвращает текущий объект истории
* window.history.go(n): Метод, позволяющий гулять по истории. В качестве аргумента передается смещение, относительно текущей позиции. Если передан 0, то будет обновлена текущая страница. Если индекс выходит за пределы истории, то ничего не произойдет.
* window.history.back(): Метод, идентичный вызову go(-1)
* window.history.forward(): Метод, идентичный вызову go(1)
* window.history.pushState(data, title [, url]): Добавляет элемент истории.
* window.history.replaceState(data, title [, url]): Обновляет текущий элемент истории

Для перехода на 2 шага назад по истории можно использовать:

history.go(-2)

Для добавления элементов истории мы можем использовать history.pushState:

history.pushState({foo: 'bar'}, 'Title', '/baz.html')

Для изменения записи истории мы можем использовать history.replaceState:

history.replaceState({foo: 'bat'}, 'New Title')

Dive into dogs это простой, но не тривиальный пример использования API истории HTML5. Он демонстрирует типичный шаблон: большая статья со связанной встроенной фотогалереей. В поддерживаемых браузерах нажатие на ссылки Next и Previous в фотогалерее будет обновлять фото в том же месте и обновлять URL в адресной строке браузера без запуска полного обновления страницы. В неподдерживаемых браузерах — или в действительности поддерживаемых браузерах, где пользователь отключил скрипты — ссылки просто работают как обычные ссылки, переводя вас на новую страницу с полным ее обновлением.

Давайте обратимся к демо и посмотрим, как оно работает. Это соответствующий код для одной фотографии.

<aside id="gallery">

<p class="photonav">

<a id="photonext" href="http://htmlbook.ru/casey.html">Next &gt;</a>

<a id="photoprev" href="http://htmlbook.ru/adagio.html">&lt; Previous</a>

</p>

<figure id="photo">

<img id="photoimg" src="http://htmlbook.ru/gallery/1972-fer-500.jpg"

alt="Fer" width="500" height="375">

<figcaption>Fer, 1972</figcaption>

</figure>

</aside>

Ничего необычного здесь нет. Фотография это <img> внутри <figure>, ссылки просто очередные элементы <a> и все завернуто в <aside>. Важно, что это всего лишь обычные ссылки, которые действительно работают. Весь код следует после скрипта проверки. Если пользователь использует неподдерживаемый браузер, наш причудливый код из API истории никогда не будет выполнен. И конечно в целом всегда есть некоторые пользователи с отключенными скриптами.

Основная функция программы получить каждую из этих ссылок и передать ее функции addClicker(), которая делает фактическую работу по созданию пользовательского обработчика click.

function setupHistoryClicks() {

addClicker(document.getElementById("photonext"));

addClicker(document.getElementById("photoprev"));

}

Это функция addClicker(). Она берет элемент <a> и добавляет обработчик click. С этим обработчиком получается интереснее.

function addClicker(link) {

link.addEventListener("click", function(e) {

swapPhoto(link.href);

history.pushState(null, null, link.href);

e.preventDefault();

}, false);

}

Функция swapPhoto() выполняет первые два шага из трех нашей трехэтапной иллюзии. В первой половине функции swapPhoto() берется часть адреса ссылки — casey.html, adagio.html и др. — и строится URL в скрытой странице, которая содержит только код, требуемый для следующей фотографии.

function swapPhoto(href) {

var req = new XMLHttpRequest();

req.open("GET",

"http://diveintohtml5.info/examples/history/gallery/" +

href.split("/").pop(),

false);

req.send(null);

Этот образец разметки возвращает http://diveintohtml5.info/examples/history/gallery/casey.html (вы можете проверить это в браузере, вставив URL напрямую).

<p class="photonav">

<a id="photonext" href="http://htmlbook.ru/brandy.html">Next &gt;</a>

<a id="photoprev" href="http://htmlbook.ru/fer.html">&lt; Previous</a>

</p>

<figure id="photo">

<img id="photoimg" src="http://htmlbook.ru/gallery/1984-casey-500.jpg"

alt="Casey" width="500" height="375">

<figcaption>Casey, 1984</figcaption>

</figure>

Это тот же основной код, что у исходной страницы, используемый для отображения первой фотографии.

Вторая половина функции swapPhoto() выполняет второй шаг нашей трехэтапной иллюзии: вставляет этот новый загруженный код в текущую страницу. Помните, что существует <aside> оборачивающий все изображение, фотографию и подпись. Вставка кода новой фотографии это шутка, достаточно установить свойство innerHtml для <aside> в свойство responseText, которое возвращается от XMLHttpRequest.

if (req.status == 200) {

document.getElementById("gallery").innerHTML = req.responseText;

setupHistoryClicks();

return true;

}

return false;

}

Также обратите внимание на вызов setupHistoryClicks(). Это необходимо, чтобы сбросить пользовательский обработчик событий click для новых вставленных ссылок. Установка innerHtml стирает любые следы старых ссылок и их обработчиков событий.

Теперь давайте вернемся к функции addClicker(). После успешной смены фотографии есть еще один шаг в нашей трехэтапной иллюзии: установить URL в адресной строке браузера без перезагрузки страницы.

history.pushState(null, null, link.href);

Функция history.pushState() содержит три параметра:

1. state может быть любой структурой данных JSON. Он передается обратно обработчику событий popstate, о котором вы узнаете чуть позже. Нам не нужно следить за state в этой демонстрации, так что я оставил его как null.
2. title может быть любой строкой. Этот параметр в настоящее время не используется основными браузерами. Если вы хотите установить заголовок страницы, вы должны сохранить его в аргументе state и установить вручную в popstate.
3. url может быть, ну, любым URL. Это URL, который должен отображаться в адресной строке браузера.

Вызов history.pushState немедленно изменит URL в адресной строке браузера. Нам еще нужно сказать о том, что происходит, когда пользователь нажимает важную кнопку «Назад».

Обычно, когда пользователь переходит на новую страницу (с полным обновлением страницы), браузер помещает новый URL в стек истории, загружает и отрисовывает новую страницу. Когда пользователь нажимает кнопку «Назад», браузер сдвигает одну страницу в стеке истории и перерисовывает предыдущую страницу. Но что происходит теперь, когда вы сделали короткое замыкание этой навигации, чтобы избежать полного обновления страницы? Итак, вы поддельно «двинулись вперед» на новый URL, так что теперь необходимо также поддельно «двинуться назад» к предыдущему URL. И ключ к поддельному «двинуться назад» в событии popstate.

window.addEventListener("popstate", function(e) {

swapPhoto(location.pathname);

}, false)

После того как вы использовали функцию history.pushState() для смещения поддельного URL в стеке истории браузера, когда пользователь нажимает кнопку «Назад», в браузере срабатывает событие popstate на объекте window. Это ваш шанс завершить иллюзию раз и навсегда. Потому что не достаточно сделать исчезновение чего-то, вы также должны вернуть его.

В этой демонстрации «вернуть его» так же просто, как смена исходной фотографии, которую мы делаем с помощью вызова swapPhoto() в текущей локации. К тому времени popstate будет вызван, URL отображается в адресной строке браузера как измененный на предыдущий URL. Кроме того, глобальное свойство location уже был обновлено с предыдущим URL.

Чтобы помочь вам представить это, давайте пройдем по шагам через всю иллюзию от начала до конца:

Пользователь загружает http://diveintohtml5.info/examples/history/fer.html, смотрит историю и фотографию Фер.

Пользователь щелкает по ссылке Next, у элемента <a> атрибут href установлен как http://diveintohtml5.info/examples/history/casey.html.

Вместо перехода на http://diveintohtml5.info/examples/history/casey.htmlс полной перезагрузкой страницы, пользовательский обработчик click на элементе <a> перехватывает щелчок и выполняет собственный код.

Наш собственный обработчик click вызывает функцию swapPhoto(), которая создает объект XMLHttpRequest для синхронной загрузки фрагмента HTML по адресу http://diveintohtml5.info/examples/history/gallery/casey.html.

Функция swapPhoto() устанавливает свойство innerHTML обертке фотогалереи (элемент <aside>), тем самым заменив фотографию Фер на фотографию Кейси.

Наконец, наш обработчик click вызывает функцию history.pushState(), чтобы вручную изменить URL в адресной строке браузера на http://diveintohtml5.info/examples/history/casey.html.

Пользователь нажимает кнопку «Назад» браузера.

Браузер замечает, что URL вручную помещается в стек истории (по функции history.pushState()). Вместо того, чтобы перейти на предыдущий URL и перерисовать всю страницу, браузер просто обновит адресную строку на предыдущий URL (http://diveintohtml5.info/examples/history/fer.html) и запустит событие popstate.

Наш пользовательский обработчик popstate снова вызовет функцию swapPhoto(), на этот раз с предыдущим URL, что сейчас уже видно в адресной строке браузера.

Снова используя XMLHttpRequest, функция swapPhoto() загружает фрагмент HTML расположенный в http://diveintohtml5.info/examples/history/gallery/fer.html и устанавливает свойство innerHtml для элемента <aside>, тем самым заменяя фотографию Кейси на фотографию Фер.

Иллюзия завершена. Все видимые доказательства (содержание страницы и URL в адресной строке) убеждают пользователя, что у него был переход вперед и назад на страницу. Но полного обновления страницы не происходит — все это было тщательно выполненной иллюзией.