

Раздел

[Сетевые запросы](#)

Навигация по уроку


Создание URL

SearchParams «?...»

Кодирование

Комментарии

Поделиться

[Редактировать на GitHub](#) → [Сетевые запросы](#) 10-го октября 2019

## Объекты URL

Встроенный класс [URL](#) предоставляет удобный интерфейс для создания и разбора URL-адресов.

Нет сетевых методов, которые требуют именно объект [URL](#), обычные строки вполне подходят. Так что, технически, мы не обязаны использовать [URL](#). Но иногда он может быть весьма удобным.

### Создание URL

Синтаксис создания нового объекта [URL](#):

```
1 new URL(url, [base])
```

- **url** – полный URL-адрес или только путь, если указан второй параметр,
- **base** – необязательный «базовый» URL: если указан и аргумент `url` содержит только путь, то адрес будет создан относительно него (пример ниже).

Например:

```
1 let url = new URL('https://javascript.info/profile/admin')
```

Эти два URL одинаковы:

```
1 let url1 = new URL('https://javascript.info/profile/admin')
2 let url2 = new URL('/profile/admin', 'https://javascript.info')
3
4 alert(url1); // https://javascript.info/profile/admin
5 alert(url2); // https://javascript.info/profile/admin
```

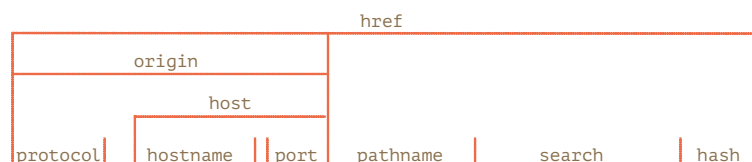
Можно легко создать новый URL по пути относительно существующего URL-адреса:

```
1 let url = new URL('https://javascript.info/profile/admin')
2 let newUrl = new URL('tester', url);
3
4 alert(newUrl); // https://javascript.info/profile/tester
```

Объект [URL](#) даёт доступ к компонентам URL, поэтому это отличный способ «разобрать» URL-адрес, например:

```
1 let url = new URL('https://javascript.info/url');
2
3 alert(url.protocol); // https:
4 alert(url.host);     // javascript.info
5 alert(url.pathname); // /url
```

Вот шпаргалка по компонентам URL:



```
https://site.com:8080/path/page?p1=v1&p2=v2...#hash
```

Раздел

[Сетевые запросы](#)

Навигация по уроку

Создание URL

SearchParams «?...»

Кодирование

Комментарии

Поделиться



Редактировать на GitHub



- href это полный URL-адрес, то же самое, что `url.toString()`
- protocol – протокол, заканчивается символом двоеточия :
- search строка параметров, начинается с вопросительного знака ?
- hash начинается с символа #
- также есть свойства `user` и `password`, если используется HTTP-аутентификация: `http://login:password@site.com` (не нарисованы сверху, так как редко используются).

**И Можно передавать объекты URL в сетевые методы (и большинство других) вместо строк**

Мы можем использовать объект URL в методах `fetch` или `XMLHttpRequest` и почти во всех других, где ожидается URL-строка.

Вообще, объект URL можно передавать почти куда угодно вместо строки, так как большинство методов сконвертируют объект в строку, при этом он станет строкой с полным URL-адресом.

## SearchParams «?...»

Допустим, мы хотим создать URL-адрес с заданными параметрами, например, `https://google.com/search?query=JavaScript`.

Мы можем указать их в строке:

```
1 new URL('https://google.com/search?query=JavaScript')
```



...Но параметры должны быть правильно закодированы, чтобы они могли содержать не-латинские буквы, пробелы и т.п. (об этом подробнее далее).



Так что для этого есть свойство `url.searchParams` – объект типа [URLSearchParams](#).

Он предоставляет удобные методы для работы с параметрами:

- `append(name, value)` – добавить параметр по имени,
- `delete(name)` – удалить параметр по имени,
- `get(name)` – получить параметр по имени,
- `getAll(name)` – получить все параметры с одинаковым именем `name` (такое возможно, например: `?user=John&user=Pete`),
- `has(name)` – проверить наличие параметра по имени,
- `set(name, value)` – задать/заменить параметр,
- `sort()` – отсортировать параметры по имени, используется редко,
- ...и является перебираемым, аналогично `Map`.

Пример добавления параметров, содержащих пробелы и знаки препинания:

```
1 let url = new URL('https://google.com/search');
2 url.searchParams.set('q', 'test me!'); // добавим парам
3
4 alert(url); // https://google.com/search?q=test+me%21
5
6 url.searchParams.set('tbs', 'qdr:y'); // параметр с дво
7
8 // параметры автоматически кодируются
9 alert(url); // https://google.com/search?query=test+me%
10
11 // перебрать параметры (в исходном виде)
12 for(let [name, value] of url.searchParams) {
13   alert(`${name}=${value}`); // q=test me!, далее tbs=q
14 }
```

Раздел

[Сетевые запросы](#)

Навигация по уроку

Создание URL

SearchParams «?...»

Кодирование

Комментарии

Поделиться



[Редактировать на GitHub](#)



## Кодирование

Существует стандарт [RFC3986](#), который определяет список разрешённых и запрещённых символов в URL.

Запрещённые символы, например, нелатинские буквы и пробелы, должны быть закодированы – заменены соответствующими кодами UTF-8 с префиксом %, например: %20 (исторически сложилось так, что пробел в URL-адресе можно также кодировать символом +, но это исключение).

К счастью, объекты URL делают всё это автоматически. Мы просто указываем параметры в обычном, незакодированном, виде, а затем конвертируем URL в строку:

```
1 let url = new URL('https://ru.wikipedia.org/wiki/Тест');
2
3 url.searchParams.set('key', 'ъ');
4 alert(url); //https://ru.wikipedia.org/wiki/%D0%A2%D0%B
```

Как видно, слово Тест в пути URL-адреса и буква ъ в параметре закодированы.

URL стал длиннее, так как каждая кириллическая буква представляется двумя байтами в кодировке UTF-8.

### Кодирование в строках

Раньше, до того как появились объекты URL, люди использовали для URL-адресов обычные строки.

Сейчас URL часто удобнее, но строки всё ещё можно использовать. Во многих случаях код с ними короче.

Однако, если мы используем строку, то надо самим позаботиться о кодировании специальных символов.

Для этого есть встроенные функции:

- [encodeURIComponent](#) – кодирует URL-адрес целиком.
- [decodeURI](#) – декодирует URL-адрес целиком.
- [encodeURIComponent](#) – кодирует компонент URL, например, параметр, хеш, имя пути и т.п.
- [decodeURIComponent](#) – декодирует компонент URL.

Возникает естественный вопрос: «Какая разница между encodeURIComponent и encodeURI? Когда использовать одну и другую функцию?»

Это легко понять, если мы посмотрим на URL-адрес, разбитый на компоненты на рисунке выше:

```
1 http://site.com:8080/path/page?p1=v1&p2=v2#hash
```

Как мы видим, в URL-адресе разрешены символы : , ? , = , & , # .

...С другой стороны, если взглянуть на один компонент, например, URL-параметр, то в нём такие символы должны быть закодированы, чтобы не поломать форматирование.

- encodeURI кодирует только символы, полностью запрещённые в URL.
- encodeURIComponent кодирует эти же символы плюс, в дополнение к ним, символы #, \$, &, +, ,, /, :, ;, =, ? и @.

Так что для URL целиком можно использовать encodeURI :

```
1 let url = encodeURI('http://site.com/привет');
2
3 alert(url); // http://site.com/%D0%BF%D1%80%D0%B8%D0%B2'
```

...А для параметров лучше будет взять `encodeURIComponent` :

```
1 let music = encodeURIComponent('Rock&Roll');
2
3 let url = `https://google.com/search?q=${music}`;
4 alert(url); // https://google.com/search?q=Rock%26Roll
```

Сравните с `encodeURI` :

```
1 let music = encodeURI('Rock&Roll');
2
3 let url = `https://google.com/search?q=${music}`;
4 alert(url); // https://google.com/search?q=Rock&Roll
```

Как видим, функция `encodeURI` не закодировала символ `&`, который является разрешённым в составе полного URL-адреса.

Но внутри параметра поиска символ `&` должен быть закодирован, в противном случае мы получим `q=Rock&Roll`, что значит `q=Rock` плюс непонятный параметр `Roll`. Не то, что предполагалось.

Чтобы правильно вставить параметр поиска в строку URL, мы должны использовать для него только `encodeURIComponent`. Наиболее безопасно кодировать и имя, и значение, за исключением случаев, когда мы абсолютно уверены в том, что они содержат только разрешённые символы.

#### Разница в кодировании с URL

Классы `URL` и `URLSearchParams` базируются на последней спецификации URI, описывающей устройство адресов: [RFC3986](#), в то время как функции `encode*` – на устаревшей версии стандарта [RFC2396](#).

Различий мало, но они есть, например, по-разному кодируются адреса IPv6:

```
1 // допустимый URL-адрес IPv6
2 let url = 'http://[2607:f8b0:4005:802::1007]/';
3
4 alert(encodeURI(url)); // http://%5B2607:f8b0:4005:802::1007%5D/
5 alert(new URL(url)); // http://[2607:f8b0:4005:802::1007]/
```

Как мы видим, функция `encodeURI` заменила квадратные скобки `[...]`, сделав адрес некорректным. Причина: URL-адреса IPv6 не существовали в момент создания стандарта RFC2396 (август 1998).

Тем не менее, такие случаи редки. По большей части функции `encode*` работают хорошо.

Проводим [курсы по JavaScript и фреймворкам](#).

## Комментарии

перед тем как писать...