

Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse


Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠 → Язык программирования JavaScript](#)
[→ Типы данных](#) 30-го ноября 2019

Формат JSON, метод toJSON

Допустим, у нас есть сложный объект, и мы хотели бы преобразовать его в строку, чтобы отправить по сети или просто вывести для логирования.

Естественно, такая строка должна включать в себя все важные свойства.

Мы могли бы реализовать преобразование следующим образом:

```
1 let user = {
2   name: "John",
3   age: 30,
4
5   toString() {
6     return `${this.name}, age: ${this.age}`;
7   }
8 };
9
10 alert(user); // {name: "John", age: 30}
```

...Но в процессе разработки добавляются новые свойства, старые свойства переименовываются и удаляются. Обновление такого `toString` каждый раз может стать проблемой. Мы могли бы попытаться перебрать свойства в нём, но что, если объект сложный, и в его свойствах имеются вложенные объекты? Мы должны были бы осуществить их преобразование тоже.

К счастью, нет необходимости писать код для обработки всего этого. У задачи есть простое решение.

JSON.stringify

JSON (JavaScript Object Notation) – это общий формат для представления значений и объектов. Его описание задокументировано в стандарте [RFC 4627](#). Первоначально он был создан для JavaScript, но многие другие языки также имеют библиотеки, которые могут работать с ним. Таким образом, JSON легко использовать для обмена данными, когда клиент использует JavaScript, а сервер написан на Ruby/PHP/Java или любом другом языке.

JavaScript предоставляет методы:

- `JSON.stringify` для преобразования объектов в JSON.
- `JSON.parse` для преобразования JSON обратно в объект.

Например, здесь мы преобразуем через `JSON.stringify` данные студента:

```
1 let student = {
2   name: 'John',
3   age: 30,
4   isAdmin: false,
5   courses: ['html', 'css', 'js'],
6   wife: null
7 };
8
9 let json = JSON.stringify(student);
10
11 alert(typeof json); // мы получили строку!
12
13
14
15
16
17
```

Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
18 alert(json);
19 /* выведет объект в формате JSON:
20 {
21   "name": "John",
22   "age": 30,
23   "isAdmin": false,
24   "courses": ["html", "css", "js"],
25   "wife": null
26 }
27 */
```

Метод `JSON.stringify(student)` берёт объект и преобразует его в строку.

Полученная строка `json` называется *JSON-форматированным* или *сериализованным* объектом. Мы можем отправить его по сети или поместить в обычное хранилище данных.

Обратите внимание, что объект в формате JSON имеет несколько важных отличий от объектного литерала:

- Строки используют двойные кавычки. Никаких одинарных кавычек или обратных кавычек в JSON. Так `'John'` становится `"John"`.
- Имена свойств объекта также заключаются в двойные кавычки. Это обязательно. Так `age:30` становится `"age":30`.

`JSON.stringify` может быть применён и к примитивам.

JSON поддерживает следующие типы данных:

- Объекты `{ ... }`
- Массивы `[...]`
- Примитивы:
 - строки,
 - числа,
 - логические значения `true/false`,
 - `null`.

Например:

```
1 // число в JSON остаётся числом
2 alert( JSON.stringify(1) ) // 1
3
4 // строка в JSON по-прежнему остаётся строкой, но в дво
5 alert( JSON.stringify('test') ) // "test"
6
7 alert( JSON.stringify(true) ); // true
8
9 alert( JSON.stringify([1, 2, 3]) ); // [1,2,3]
```

JSON является независимой от языка спецификацией для данных, поэтому `JSON.stringify` пропускает некоторые специфические свойства объектов JavaScript.

А именно:

- Свойства-функции (методы).
- Символьные свойства.
- Свойства, содержащие `undefined`.

```
1 let user = {
2   sayHi() { // будет пропущено
3     alert("Hello");
4   },
5   [Symbol("id")]: 123, // также будет пропущено
6   something: undefined // как и это - пропущено
7 };
8
9 alert( JSON.stringify(user) ); // {} (пустой объект)
```

Раздел

Типы данных

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



Обычно это нормально. Если это не то, чего мы хотим, то скоро мы увидим, как можно настроить этот процесс.

Самое замечательное, что вложенные объекты поддерживаются и конвертируются автоматически.

Например:

```
1 let meetup = {
2   title: "Conference",
3   room: {
4     number: 23,
5     participants: ["john", "ann"]
6   }
7 };
8
9 alert( JSON.stringify(meetup) );
10 /* вся структура преобразована в строку:
11 {
12   "title":"Conference",
13   "room":{"number":23,"participants":["john","ann"]},
14 }
15 */
```



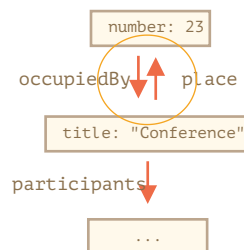
Важное ограничение: не должно быть циклических ссылок.

Например:

```
1 let room = {
2   number: 23
3 };
4
5 let meetup = {
6   title: "Conference",
7   participants: ["john", "ann"]
8 };
9
10 meetup.place = room;           // meetup ссылается на room
11 room.occupiedBy = meetup;      // room ссылается на meetup
12
13 JSON.stringify(meetup); // Ошибка: Преобразование цикла
```



Здесь преобразование завершается неудачно из-за циклической ссылки: room.occupiedBy ссылается на meetup, и meetup.place ссылается на room:



Исключаем и преобразуем: replacer

Полный синтаксис JSON.stringify:

```
1 let json = JSON.stringify(value[, replacer, space])
```

value

Значение для кодирования.

replacer

Раздел

Типы данных

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



Массив свойств для кодирования или функция соответствия
`function(key, value)`.

space

Дополнительное пространство (отступы), используемое для форматирования.

В большинстве случаев `JSON.stringify` используется только с первым аргументом. Но если нам нужно настроить процесс замены, например, отфильтровать циклические ссылки, то можно использовать второй аргумент `JSON.stringify`.

Если мы передадим ему массив свойств, будут закодированы только эти свойства.

Например:

```
1 let room = {
2   number: 23
3 };
4
5 let meetup = {
6   title: "Conference",
7   participants: [{name: "John"}, {name: "Alice"}],
8   place: room // meetup ссылается на room
9 };
10
11 room.occupiedBy = meetup; // room ссылается на meetup
12
13 alert( JSON.stringify(meetup, ['title', 'participants'])
14 // {"title":"Conference","participants":[{"name":"John"}, {"name":"Alice"}]}
```

Здесь мы, наверное, слишком строги. Список свойств применяется ко всей структуре объекта. Так что внутри `participants` – пустые объекты, потому что `name` нет в списке.

Давайте включим в список все свойства, кроме `room.occupiedBy`, из-за которого появляется циклическая ссылка:

```
1 let room = {
2   number: 23
3 };
4
5 let meetup = {
6   title: "Conference",
7   participants: [{name: "John"}, {name: "Alice"}],
8   place: room // meetup ссылается на room
9 };
10
11 room.occupiedBy = meetup; // room ссылается на meetup
12
13 alert( JSON.stringify(meetup, ['title', 'participants'],
14 /*
15 {
16   "title":"Conference",
17   "participants":[{"name":"John"}, {"name":"Alice"}],
18   "place":{"number":23}
19 }
20 */
```

Теперь всё, кроме `occupiedBy`, сериализовано. Но список свойств довольно длинный.

К счастью, в качестве `replacer` мы можем использовать функцию, а не массив.

Функция будет вызываться для каждой пары (`key`, `value`), и она должна возвращать заменённое значение, которое будет использоваться вместо исходного. Или `undefined`, чтобы пропустить значение.

Раздел

Типы данных

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



В нашем случае мы можем вернуть value «как есть» для всего, кроме occupiedBy. Чтобы игнорировать occupiedBy, код ниже возвращает undefined:

```
1 let room = {
2   number: 23
3 };
4
5 let meetup = {
6   title: "Conference",
7   participants: [{name: "John"}, {name: "Alice"}],
8   place: room // meetup ссылается на room
9 };
10
11 room.occupiedBy = meetup; // room ссылается на meetup
12
13 alert( JSON.stringify(meetup, function replacer(key, va
14   alert(`${key}: ${value}`);
15   return (key == 'occupiedBy') ? undefined : value;
16 }));
17
18 /* пары ключ:значение, которые приходят в replacer:
19 :           [object Object]
20 title:      Conference
21 participants: [object Object],[object Object]
22 0:          [object Object]
23 name:       John
24 1:          [object Object]
25 name:       Alice
26 place:      [object Object]
27 number:     23
28 */
```



Обратите внимание, что функция replacer получает каждую пару ключ/значение, включая вложенные объекты и элементы массива. И она применяется рекурсивно. Значение this внутри replacer – это объект, который содержит текущее свойство.

Первый вызов – особенный. Ему передаётся специальный «объект-обёртка»: {"": meetup}. Другими словами, первая (key, value) пара имеет пустой ключ, а значением является целевой объект в общем. Вот почему первая строка из примера выше будет ": [object Object]".

Идея состоит в том, чтобы дать как можно больше возможностей replacer – у него есть возможность проанализировать и заменить/пропустить даже весь объект целиком, если это необходимо.

Форматирование: space

Третий аргумент в JSON.stringify(value, replacer, space) – это количество пробелов, используемых для удобного форматирования.

Ранее все JSON-форматированные объекты не имели отступов и лишних пробелов. Это нормально, если мы хотим отправить объект по сети. Аргумент space используется исключительно для вывода в удобочитаемом виде.

Ниже space = 2 указывает JavaScript отображать вложенные объекты в несколько строк с отступом в 2 пробела внутри объекта:

```
1 let user = {
2   name: "John",
3   age: 25,
4   roles: {
5     isAdmin: false,
6     isEditor: true
7   }
8 };
9
10 alert(JSON.stringify(user, null, 2));
```



Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
11  /* отступ в 2 пробела:
12  {
13    "name": "John",
14    "age": 25,
15    "roles": {
16      "isAdmin": false,
17      "isEditor": true
18    }
19  }
20  */
21
22  /* для JSON.stringify(user, null, 4) результат содержит
23  {
24    "name": "John",
25    "age": 25,
26    "roles": {
27      "isAdmin": false,
28      "isEditor": true
29    }
30  }
31  */
```

Параметр space применяется для логирования и красивого вывода.

Пользовательский «toJSON»

Как и toString для преобразования строк, объект может предоставлять метод toJSON для преобразования в JSON. JSON.stringify автоматически вызывает его, если он есть.

Например:

```
1  let room = {
2    number: 23
3  };
4
5  let meetup = {
6    title: "Conference",
7    date: new Date(Date.UTC(2017, 0, 1)),
8    room
9  };
10
11 alert( JSON.stringify(meetup) );
12 /*
13  {
14    "title": "Conference",
15    "date": "2017-01-01T00:00:00.000Z", // (1)
16    "room": {"number": 23}           // (2)
17  }
18  */
```

Как видим, date (1) стал строкой. Это потому, что все объекты типа Date имеют встроенный метод toJSON, который возвращает такую строку.

Теперь давайте добавим собственную реализацию метода toJSON в наш объект room (2):

```
1  let room = {
2    number: 23,
3    toJSON() {
4      return this.number;
5    }
6  };
7
8  let meetup = {
9    title: "Conference",
10   room
11  };
```

Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
12
13 alert( JSON.stringify(room) ); // 23
14
15 alert( JSON.stringify(meetup) );
16 /*
17   {
18     "title": "Conference",
19     "room": 23
20   }
21  */
```

Как видите, `toJSON` используется как при прямом вызове `JSON.stringify(room)`, так и когда `room` вложен в другой сериализуемый объект.

JSON.parse

Чтобы декодировать JSON-строку, нам нужен другой метод с именем [JSON.parse](#).

Синтаксис:

```
1 let value = JSON.parse(str, [reviver]);
```

str

JSON для преобразования в объект.

reviver

Необязательная функция, которая будет вызываться для каждой пары (ключ, значение) и может преобразовывать значение.

Например:

```
1 // строковый массив
2 let numbers = "[0, 1, 2, 3]";
3
4 numbers = JSON.parse(numbers);
5
6 alert( numbers[1] ); // 1
```

Или для вложенных объектов:

```
1 let user = '{ "name": "John", "age": 35, "isAdmin": false }';
2
3 user = JSON.parse(user);
4
5 alert( user.friends[1] ); // 1
```

JSON может быть настолько сложным, насколько это необходимо, объекты и массивы могут включать другие объекты и массивы. Но они должны быть в том же JSON-формате.

Вот типичные ошибки в написанном от руки JSON (иногда приходится писать его для отладки):

```
1 let json = `{
2   name: "John", // Ошибка: имя свой
3   "surname": 'Smith', // Ошибка: одинарные
4   'isAdmin': false // Ошибка: одинарные
5   "birthday": new Date(2000, 2, 3), // Ошибка: не допус
6   "friends": [0,1,2,3] // Здесь всё
7 }`;
```

Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Кроме того, JSON не поддерживает комментарии. Добавление комментария в JSON делает его недействительным.

Существует ещё один формат [JSON5](#), который поддерживает ключи без кавычек, комментарии и т.д. Но это самостоятельная библиотека, а не спецификация языка.

Обычный JSON настолько строг не потому, что его разработчики ленивы, а потому, что позволяет легко, надёжно и очень быстро реализовывать алгоритм кодирования и чтения.

Использование reviver

Представьте, что мы получили объект `meetup` с сервера в виде строки данных.

Вот такой:

```
1 // title: (meetup title), date: (meetup date)
2 let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';
```

...А теперь нам нужно *десериализовать* её, т.е. снова превратить в объект JavaScript.

Давайте сделаем это, вызвав `JSON.parse`:

```
1 let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';
2
3 let meetup = JSON.parse(str);
4
5 alert( meetup.date.getDate() ); // Error!
```

Ой, ошибка!

Значением `meetup.date` является строка, а не `Date` объект. Как `JSON.parse` мог знать, что он должен был преобразовать эту строку в `Date`?

Давайте передадим `JSON.parse` функцию восстановления вторым аргументом, которая возвращает все значения «как есть», но `date` станет `Date`:

```
1 let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';
2
3 let meetup = JSON.parse(str, function(key, value) {
4   if (key == 'date') return new Date(value);
5   return value;
6 });
7
8 alert( meetup.date.getDate() ); // 30 - теперь работает
```

Кстати, это работает и для вложенных объектов:

```
1 let schedule = `{
2   "meetups": [
3     {"title":"Conference","date":"2017-11-30T12:00:00.000Z"},
4     {"title":"Birthday","date":"2017-04-18T12:00:00.000Z"}
5   ]
6 `;
7
8 schedule = JSON.parse(schedule, function(key, value) {
9   if (key == 'date') return new Date(value);
10  return value;
11 });
12
13 alert( schedule.meetups[1].date.getDate() ); // 18 - от.
```


Раздел

[Типы данных](#)

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



[Редактировать на GitHub](#)

Итого

- JSON – это формат данных, который имеет собственный независимый стандарт и библиотеки для большинства языков программирования.
- JSON поддерживает простые объекты, массивы, строки, числа, логические значения и `null`.
- JavaScript предоставляет методы `JSON.stringify` для сериализации в JSON и `JSON.parse` для чтения из JSON.
- Оба метода поддерживают функции преобразования для интеллектуального чтения/записи.
- Если объект имеет метод `toJSON`, то он вызывается через `JSON.stringify`.

✓ Задачи

Преобразуйте объект в JSON, а затем обратно в обычный объект

важность: 5

Преобразуйте `user` в JSON, затем прочитайте этот JSON в другую переменную.

```
1 let user = {  
2   name: "Василий Иванович",  
3   age: 35  
4 };;
```

решение

Исключить обратные ссылки

важность: 5

В простых случаях циклических ссылок мы можем исключить свойство, из-за которого они возникают, из сериализации по его имени.

Но иногда мы не можем использовать имя, так как могут быть и другие, нужные, свойства с этим именем во вложенных объектах. Поэтому можно проверять свойство по значению.

Напишите функцию `replacer` для JSON-преобразования, которая удалит свойства, ссылающиеся на `meetup`:

```
1 let room = {  
2   number: 23  
3 };  
4  
5 let meetup = {  
6   title: "Совещание",  
7   occupiedBy: [{name: "Иванов"}, {name: "Петров"}],  
8   place: room  
9 };  
10  
11 // циклические ссылки  
12 room.occupiedBy = meetup;  
13 meetup.self = meetup;  
14  
15 alert( JSON.stringify(meetup, function replacer(key, va  
16   /* ваш код */  
17 ));  
18  
19 /* в результате должно быть:  
20 {  
21   "title": "Совещание",  
22   "occupiedBy": [{ "name": "Иванов" }, { "name": "Петров" } ],  
23   "place": { "number": 23 }
```

24 }
25 */

Раздел

Типы данных

Навигация по уроку

JSON.stringify

Исключаем и преобразуем:
replacer

Форматирование: space

Пользовательский «toJSON»

JSON.parse

Использование reviver

Итого

Задачи (2)

Комментарии

Поделиться



Редактировать на GitHub



решение



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



Комментарии

перед тем как писать...

© 2007–2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

