Основы JS

Работа с файлами и JSON

Формат JSON

Формат JSON используется для представления объектов в виде строки.

Зачем?

- Для удобства хранения объектов.
- Если нужно с сервера взять объект с данными и передать его клиенту, то в качестве промежуточного формата – для передачи по сети, почти всегда используют именно его.

Формат JSON

Основные методы для работы с JSON в JavaScript:

- `JSON.parse` – читает объекты из строки в формате JSON.
- `JSON.stringify` – превращает объекты в строку в формате JSON, используется, когда нужно из JavaScript передать данные по сети.

Данные в формате JSON (RFC 4627) представляют собой:

- JavaScript-объекты { ... } или
- Массивы [...] или
- Значения одного из типов:
  - строки в двойных кавычках,
  - число,
  - логическое значение true/false,
  - null.

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```

Формат JSON. Особенности

JSON-объекты ≠ JavaScript-объекты

- Объекты в формате JSON похожи на обычные JavaScript-объекты, но отличаются от них более строгими требованиями к строкам – они должны быть именно в двойных кавычках.
- В формате JSON не поддерживаются комментарии. Он предназначен только для передачи данных.
- JSON - это чисто формат данных - он содержит только свойства, а не методы.
- Даже одна неуместная запятая или двоеточие может привести к сбою JSON-файла и не работать.

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```

Работа с файловой системой

NodeJS предоставляет библиотеку fs для работы с файловой системой

Чтение данных может быть:

- асинхронным (фишка JS - можно работать с несколькими запросами чтения параллельно)
- синхронным (один запрос чтения, похоже на реализацию чтения в C++, python и т.д.)

Асинхронное и синхронное чтение

examples/node/non-blocking-read-file.js

```
1. var fs = require('fs');
2.
3. fs.readFile('DATA', 'utf8', function(err, contents) {
4.     console.log(contents);
5. });
6.
7. console.log('after calling readFile');
8.
```

examples/node/blocking-read-file.js

```
1. var fs = require('fs');
2.
3. var contents = fs.readFileSync('DATA', 'utf8');
4. console.log(contents);
5.
```

Открытие файла

```
fs.open(path, flags, callback)
```

Описание используемых параметров:

path — это строка с именем файла, включая путь к нему.

flags — указывают режим работы с файлом, который нужно открыть. Все возможные значения приведены ниже.

callback — это функция обратного вызова, которая принимает два аргумента (err, fd).

Запись файла

```
fs.writeFile(filename, data[,options], callback)
```

Описание используемых параметров:

path — это строка с именем файла, включая путь к нему.

data – это строка или буфер, которые должны быть записаны в файл.

options — Третий параметр — это объект, который содержит {encoding, mode, flag}. По умолчанию encoding — utf8, mode — восьмеричное значение 0666 и flag — w,

callback — это функция обратного вызова, которая получает единственный параметр err, который возвращает ошибку в случае возникновения ошибки записи.

Запись файла. Пример

`fs.writeFile(filename, data[,options], callback)`

```
fs.writeFile('input.txt', 'JS Learning!',  function(err) {
    if (err) {
        return console.error(err);
    }

    console.log("Запись добавлена!");
    console.log("Чтение из файла");
    fs.readFile('input.txt', function (err, data) {
        if (err) {
            return console.error(err);
        }
        console.log("Asynchronous read: " + data.toString());
    });
});
```

Спасибо!

Ссылка на материалы урока:
https://github.com/akenoq/ing-prog-web-1