
# IndexedDB

# ЧТО ТАКОЕ INDEXEDDB?

**IndexedDB** – это встроенная база данных на стороне клиента, более мощная, чем localStorage.
Особенности IndexedDB:

- Хранилище ключей/значений: доступны несколько типов ключей, а значения могут быть (почти) любыми.
- Поддерживает транзакции для надёжности.
- Поддерживает запросы в диапазоне ключей и индексы.
- Позволяет хранить больше данных, чем localStorage.

IndexedDB предназначена в основном для оффлайн приложений, для обычных клиент-серверных приложений этот функционал избыточен.

# ОТКРЫТЬ БАЗУ ДАННЫХ

Для начала работы с IndexedDB нужно открыть базу данных.

```
let name = "todo_list"; //имя БД
let version = 2; //Версия БД
let openRequest = indexedDB.open(name, version);
```

По результатам открытия могут сработать три события:

- onupgradeneeded;
- onerror;
- onsuccess.

# ОТКРЫТЬ БАЗУ ДАННЫХ

```
openRequest.onupgradeneeded = function(event) {
        // срабатывает, если на клиенте нет базы данных
        // срабатывает, если отличается версия базы данных
        // ...выполнить инициализацию базы...
        let db = openRequest.result;
        switch(db. oldVersion) { /* существующая(старая) версия базы данных */
            case 0:
                        // версия 0 означает, что на клиенте нет базы данных
                        // выполнить инициализацию
                        console.log("Базы данных не существует");
                        break;
            case 1:
                        // на клиенте версия базы данных 1
                        // обновить
                        console.log("Базы данных устаревшей версии");
        }
        /*после этого обработчика будет запущено событие onsuccess, так как
считается что база обновлена*/
};
```

# ОТКРЫТЬ БАЗУ ДАННЫХ

```
openRequest.onerror = function() {
    console.error("Error", openRequest.error);
    //например попытка открыть базу данных
    //с более низкой версией, чем текущая
};

openRequest.onsuccess = function() {
    //успешное открытие базы данных
    let db = openRequest.result;
    // продолжить работу с бд, используя объект db
};
```

# УДАЛЕНИЕ БАЗЫ ДАННЫХ

```
let name = "todo_list_bd"; //имя БД
let deleteRequest =
indexedDB.deleteDatabase(name);

deleteRequest.onsuccess = function(){
        //Успешное удаление базы данных
        console.log("База данных удалена");
};

deleteRequest.onerror  = function(){
        //Ошибка удаления
        console.error("Error", deleteRequest.error);
};
```

# ПАРАЛЛЕЛЬНОЕ ОБНОВЛЕНИЕ БД

Ситуация:

- Допустим, посетитель открыл наш сайт во вкладке браузера, с базой версии 1.
- Затем мы выкатили обновление, и тот же посетитель открыл наш сайт в другой вкладке. Так что есть две вкладки, на которых открыт наш сайт, но в одной открыто соединение с базой версии 1, а другая пытается обновить версию базы в обработчике upgradeneeded.
- Проблема заключается в том, что база данных всего одна на две вкладки, так как это один и тот же сайт, один источник. И она не может быть одновременно версии 1 и 2. Чтобы обновить на версию 2, все соединения к версии 1 должны быть закрыты.

# ПАРАЛЛЕЛЬНОЕ ОБНОВЛЕНИЕ БД

```
openRequest.onsuccess = function() {
    let db = openRequest.result;
    // ...база данных доступна как объект db...
    db.onversionchange = function() {
        //событие сработает при попытке параллельного обновления
        db.close();
        alert("База данных устарела, пожалуйста, перезагрузите страницу.")
    };
  };

openRequest.onblocked = function() {
    /* есть другое соединение к той же базе и оно не было закрыто после срабатывания на нём db.onversionchange */
    alert("Пожалуйста, перезагрузите другие вкладки работающие с нашим сайтом.");
};
```

# ХРАНИЛИЩЕ ОБЪЕКТОВ

Хранилище объектов – это основная концепция IndexedDB. В других базах данных это «таблицы» или «коллекции».

Хранилище объектов можно создавать/изменять/удалять только при обновлении версии базы данных в обработчике upgradeneeded.

Каждому значению в хранилище должен соответствовать уникальный ключ.

Хранилище объектов внутренне сортирует значения по ключам.

Можно указать ключ при добавлении значения в хранилище, аналогично localStorage. Но когда мы храним объекты, IndexedDB позволяет установить свойство объекта в качестве ключа, что гораздо удобнее. Или мы можем автоматически сгенерировать ключи.

# ХРАНИЛИЩЕ ОБЪЕКТОВ

Создание хранилища:
db.createObjectStore(name[, keyOptions]);

- name – имя хранилища
- keyOptions – это необязательный объект с одним или двумя свойствами:
  - keyPath – путь к свойству объекта, которое IndexedDB будет использовать в качестве ключа, например id.
  - autoIncrement – если true, то ключ будет формироваться автоматически для новых объектов, как постоянно увеличивающееся число.

```
/* Например, создаться хранилище объектов использующих свойство id как ключ: */
db.createObjectStore('todos', {keyPath: 'id'});
```

# ХРАНИЛИЩЕ ОБЪЕКТОВ

Проверка существования хранилища:
db.objectStoreNames.contains(name)

- name – имя хранилища

db.objectStoreNames – содержит DOMStringList со вмесим хранилищами базы данных.

Чтобы удалить хранилище объектов:
db.deleteObjectStore(name);

- name – имя хранилища

Пример:

```
if (!db.objectStoreNames.contains('todos')) {
      db.createObjectStore('todos_list', {keyPath: 'id'});
      db.deleteObjectStore('todos');
}
```

# ТРАНЗАКЦИИ

Транзакция – это группа операций, которые должны быть или все выполнены, или все не выполнены.

Все операции с данными в IndexedDB могут быть сделаны только внутри транзакций.

Транзакции завершаются самостоятельно по завершению блока кода, т.е. мы не можем вставить асинхронную операцию, такую как например setTimeout в середину транзакции.

Для начала транзакции:
db.transaction(store[, type]);

- store – это название хранилища, к которому транзакция получит доступ. Может быть массивом названий, если нам нужно предоставить доступ к нескольким хранилищам.
- type – тип транзакции, один из:
  - readonly – только чтение, по умолчанию.
  - readwrite – только чтение и запись данных.

# ТРАНЗАКЦИИ

Хранилища объектов поддерживают два метода для добавления значений:

- put(value, [key]) - Добавляет значение value в хранилище. Ключ key необходимо указать, если при создании хранилища объектов не было указано свойство keyPath или autoIncrement. Если уже есть значение с таким же ключом, то оно будет заменено.
- add(value, [key]) То же, что put, но если уже существует значение с таким ключом, то запрос не выполнится, будет сгенерирована ошибка с названием "ConstraintError".

Аналогично открытию базы, мы отправляем запрос: books.add(book) и после ожидаем события success/error. request.result для add является ключом нового объекта. Ошибка находится в request.error (если есть).

# ТРАНЗАКЦИИ

Пример:

```
 /*создать транзакцию и указать все хранилища, к которым необходим доступ*/
let transaction = db.transaction("todos", "readwrite");
// получить хранилище объектов для работы с ним
let todos = transaction.objectStore("todos");

let todo = { id: 1, task: 'task 1', created: new Date() };
//Выполнить запрос на добавление элемента в хранилище объектов
let request = todos.add(todo);

request.onsuccess = function() {
        console.log("Задача добавлена в хранилище", request.result);
};

request.onerror = function() {
    console.log("Ошибка", request.error);
};
```

# ПОИСК ПО КЛЮЧАМ

Есть два основных вида поиска в хранилище объектов:

- По ключу или по диапазону ключей.
- По полям объекта.

Диапазоны создаются с помощью следующих вызовов:

- IDBKeyRange.lowerBound(lower, [open]) означает: >lower (или ≥lower, если open это true)
- IDBKeyRange.upperBound(upper, [open]) означает: <upper (или ≤upper, если open это true)
- IDBKeyRange.bound(lower, upper, [lowerOpen], [upperOpen]) означает: между lower и upper, включительно, если соответствующий lowerOpen и upperOpen равен true.
- IDBKeyRange.only(key) – диапазон, который состоит только из одного ключа key, редко используется.

# ПОИСК ПО КЛЮЧАМ

Все методы поиска принимают аргумент query, который может быть либо точным ключом, либо диапазоном ключей:

- store.get(query) – поиск первого значения по ключу или по диапазону.
- store.getAll([query], [count]) – поиск всех значений, можно ограничить, передав count.
- store.getKey(query) – поиск первого ключа, который удовлетворяет запросу, обычно передаётся диапазон.
- store.getAllKeys([query], [count]) – поиск всех ключей, которые удовлетворяют запросу, обычно передаётся диапазон, возможно ограничить поиск, передав count.
- store.count([query]) – получить общее количество ключей, которые удовлетворяют запросу, обычно передаётся диапазон.

# ПОИСК ПО КЛЮЧАМ

Пример:

```
//Создаем транзацию
let transaction = db.transaction("todos", "readwrite");
//Получаем объект хранилище
let todos = transaction.objectStore("todos");
//Формируем запрос на поиск задачи с ключом 1
let request = todos.get(1);

request.onsuccess = function() {
    console.log("Считана задача из хранилища", request.result);
};

request.onerror = function() {
    console.log("Ошибка", request.error);
};
```

# ПОИСК ПО ИНДЕКСИРОВАННОМУ ПОЛЮ

Для поиска по другим полям объекта нам нужно создать дополнительную структуру данных, называемую «индекс» (index). Индексы должны создаваться в upgradeneeded, как и хранилище объектов.

Синтаксис:
objectStore.createIndex(name, keyPath, [options]);

- name – название индекса.
- keyPath – путь к полю объекта, которое индекс должен отслеживать.
- option – необязательный объект со свойствами:
  - unique – если true, тогда в хранилище может быть только один объект с заданным значением в keyPath. Если мы попытаемся добавить дубликат, то индекс сгенерирует ошибку.
  - multiEntry – используется только, если keyPath является массивом. В этом случае, по умолчанию, индекс обрабатывает весь массив как ключ. Но если мы укажем true в multiEntry, тогда индекс будет хранить список объектов хранилища для каждого значения в этом массиве. Таким образом, элементы массива становятся ключами индекса.

# ПОИСК ПО ИНДЕКСИРОВАННОМУ ПОЛЮ

Пример:

```
//в обработчике onupgradeneeded
let todos = db.createObjectStore('todos', {keyPath: 'id'});
todos.createIndex('data_idx', 'created');

//в обработчике onsuccess
//Создаем транзацию
let transaction = db.transaction("todos", "readwrite");
//Получаем объект хранилище
let todos = transaction.objectStore("todos");
//Формируем запрос на поиск
let request = todos
    .index("data_idx")
    .getAll(IDBKeyRange.upperBound(new Date, true));

request.onsuccess = function() {
    console.log("Считана задача из хранилища", request.result);
};
request.onerror = function() { console.log("Ошибка", request.error); };
```

# УДАЛЕНИЕ ИЗ ХРАНИЛИЩА

Метод delete удаляет значения по запросу query:
delete(query) - аргумент query, который может быть либо точным ключом, либо диапазоном ключей. Удаление идёт только по ключу.
clear() – очищает хранилище.
Пример:

```
//Создаем транзацию
let transaction = db.transaction("todos", "readwrite");
//Получаем объект хранилище
let todos = transaction.objectStore("todos");
//Формируем запрос на удлание
let request = todos.delete(1);
request.onsuccess = function() {
    console.log("Удалена задача с id = 1 из хранилища");
};

request.onerror = function() {
    console.log("Ошибка", request.error);
};
```

# ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

https://learn.javascript.ru/indexeddb - IndexedDB (онлайн учебник)

https://www.w3.org/TR/IndexedDB/ - Спецификация Indexed Database API 2.0 (на английском языке)