IndexedDB API

概述

随着浏览器的功能不断增强，越来越多的网站开始考虑，将大量数据储存在客户端，这样可以减少从服务器获取数据，直接从本地获取数据。

现有的浏览器数据储存方案，都不适合储存大量数据：Cookie 的大小不超过 4KB，且每次请求都会发送回服务器；LocalStorage 在 2.5MB 到 10MB 之间（各家浏览器不同），而且不提供搜索功能，不能建立自定义的索引。所以，需要一种新的解决方案，这就是 IndexedDB 诞生的背景。

通俗地说，IndexedDB 就是浏览器提供的本地数据库，它可以被网页脚本创建和操作。IndexedDB 允许储存大量数据，提供查找接口，还能建立索引。这些都是 LocalStorage 所不具备的。就数据库类型而言，IndexedDB 不属于关系型数据库（不支持 SQL 查询语句），更接近 NoSQL 数据库。

IndexedDB 具有以下特点。

（1）键值对储存。 IndexedDB 内部采用对象仓库（object store）存放数据。所有类型的数据都可以直接存入，包括 JavaScript 对象。对象仓库中，数据以“键值对”的形式保存，每一个数据记录都有对应的主键，主键是独一无二的，不能有重复，否则会抛出一个错误。

（2）异步。 IndexedDB 操作时不会锁死浏览器，用户依然可以进行其他操作，这与 LocalStorage 形成对比，后者的操作是同步的。异步设计是为了防止大量数据的读写，拖慢网页的表现。

（3）支持事务。 IndexedDB 支持事务（transaction），这意味着一系列操作步骤之中，只要有一步失败，整个事务就都取消，数据库回滚到事务发生之前的状态，不存在只改写一部分数据的情况。

（4）同源限制。 IndexedDB 受到同源限制，每一个数据库对应创建它的域名。网页只能访问自身域名下的数据库，而不能访问跨域的数据库。

（5）储存空间大。 IndexedDB 的储存空间比 LocalStorage 大得多，一般来说不少于 250MB，甚至没有上限。

（6）支持二进制储存。 IndexedDB 不仅可以储存字符串，还可以储存二进制数据（ArrayBuffer 对象和 Blob 对象）。

基本概念

IndexedDB 是一个比较复杂的 API，涉及不少概念。它把不同的实体，抽象成一个个对象接口。学习这个 API，就是学习它的各种对象接口。

- 数据库：IDBDatabase 对象
- 对象仓库：IDBObjectStore 对象
- 索引： IDBIndex 对象
- 事务： IDBTransaction 对象
- 操作请求：IDBRequest 对象
- 指针： IDBCursor 对象
- 主键集合：IDBKeyRange 对象

下面是一些主要的概念。

（1）数据库

数据库是一系列相关数据的容器。每个域名（严格的说，是协议 + 域名 + 端口）都可以新建任意多个数据库。

IndexedDB 数据库有版本的概念。同一个时刻，只能有一个版本的数据库存在。如果要修改数据库结构（新增或删除表、索引或者主键），只能通过升级数据库版本完成。

（2）对象仓库

每个数据库包含若干个对象仓库（object store）。它类似于关系型数据库的表格。

（3）数据记录

对象仓库保存的是数据记录。每条记录类似于关系型数据库的行，但是只有主键和数据体两部分。主键用来建立默认的索引，必须是不同的，否则会报错。主键可以是数据记录里面的一个属性，也可以指定为一个递增的整数编号。

```javascript
{ id: 1, text: 'foo' }
```

上面的对象中，`id`属性可以当作主键。

数据体可以是任意数据类型，不限于对象。

（4）索引

为了加速数据的检索，可以在对象仓库里面，为不同的属性建立索引。

（5）事务

数据记录的读写和删改，都要通过事务完成。事务对象提供`error`、`abort`和`complete`三个事件，用来监听操作结果。

操作流程

IndexedDB 数据库的各种操作，一般是按照下面的流程进行的。这个部分只给出简单的代码示例，用于快速上手，详细的各个对象的 API 放在后文介绍。

打开数据库

使用 IndexedDB 的第一步是打开数据库，使用`indexedDB.open()`方法。

```javascript
var request = window.indexedDB.open(databaseName, version);
```

这个方法接受两个参数，第一个参数是字符串，表示数据库的名字。如果指定的数据库不存在，就会新建数据库。第二个参数是整数，表示数据库的版本。如果省略，打开已有数据库时，默认为当前版本；新建数据库时，默认为`1`。

`indexedDB.open()`方法返回一个 IDBRequest 对象。这个对象通过三种事件`error`、`success`、`upgradeneeded`，处理打开数据库的操作结果。

（1）error 事件

`error`事件表示打开数据库失败。

```javascript
request.onerror = function (event) {
  console.log('数据库打开报错');
};
```

（2）success 事件

`success`事件表示成功打开数据库。

```javascript
var db;

request.onsuccess = function (event) {
  db = request.result;
  console.log('数据库打开成功');
};
```

这时，通过`request`对象的`result`属性拿到数据库对象。

（3）upgradeneeded 事件

如果指定的版本号，大于数据库的实际版本号，就会发生数据库升级事件`upgradeneeded`。

```javascript
var db;

request.onupgradeneeded = function (event) {
  db = event.target.result;
}
```

这时通过事件对象的`target.result`属性，拿到数据库实例。

新建数据库

新建数据库与打开数据库是同一个操作。如果指定的数据库不存在，就会新建。不同之处在于，后续的操作主要在`upgradeneeded`事件的监听函数里面完成，因为这时版本从无到有，所以会触发这个事件。

通常，新建数据库以后，第一件事是新建对象仓库（即新建表）。

```javascript
request.onupgradeneeded = function(event) {
  db = event.target.result;
  var objectStore = db.createObjectStore('person', { keyPath: 'id' });
}
```

上面代码中，数据库新建成功以后，新增一张叫做`person`的表格，主键是`id`。

更好的写法是先判断一下，这张表格是否存在，如果不存在再新建。

```javascript
request.onupgradeneeded = function (event) {
  db = event.target.result;
  var objectStore;
  if (!db.objectStoreNames.contains('person')) {
    objectStore = db.createObjectStore('person', { keyPath: 'id' });
  }
}
```

主键（key）是默认建立索引的属性。比如，数据记录是`{ id: 1, name: '张三' }`，那么`id`属性可以作为主键。主键也可以指定为下一层对象的属性，比如`{ foo: { bar: 'baz' } }`的`foo.bar`也可以指定为主键。

如果数据记录里面没有合适作为主键的属性，那么可以让 IndexedDB 自动生成主键。

```javascript
var objectStore = db.createObjectStore(
  'person',
  { autoIncrement: true }
);
```

上面代码中，指定主键为一个递增的整数。

新建对象仓库以后，下一步可以新建索引。

```javascript
request.onupgradeneeded = function(event) {
  db = event.target.result;
  var objectStore = db.createObjectStore('person', { keyPath: 'id' });
  objectStore.createIndex('name', 'name', { unique: false });
  objectStore.createIndex('email', 'email', { unique: true });
}
```

上面代码中，`IDBObject.createIndex()`的三个参数分别为索引名称、索引所在的属性、配置对象（说明该属性是否包含重复的值）。

新增数据

新增数据指的是向对象仓库写入数据记录。这需要通过事务完成。

```javascript
function add() {
  var request = db.transaction(['person'], 'readwrite')
    .objectStore('person')
    .add({ id: 1, name: '张三', age: 24, email: 'zhangsan@example.com' });

  request.onsuccess = function (event) {
    console.log('数据写入成功');
  };

  request.onerror = function (event) {
    console.log('数据写入失败');
  }
}

add();
```

上面代码中，写入数据需要新建一个事务。新建时必须指定表格名称和操作模式（“只读”或“读写”）。新建事务以后，通过`IDBTransaction.objectStore(name)`方法，拿到 IDBObjectStore 对象，再通过表格对象的`add()`方法，向表格写入一条记录。

写入操作是一个异步操作，通过监听连接对象的`success`事件和`error`事件，了解是否写入成功。

读取数据

读取数据也是通过事务完成。

```javascript
function read() {
  var transaction = db.transaction(['person']);
  var objectStore = transaction.objectStore('person');
  var request = objectStore.get(1);

  request.onerror = function(event) {
    console.log('事务失败');
  };

  request.onsuccess = function( event) {
    if (request.result) {
      console.log('Name: ' + request.result.name);
      console.log('Age: ' + request.result.age);
      console.log('Email: ' + request.result.email);
    } else {
      console.log('未获得数据记录');
    }
  };
}

read();
```

上面代码中，`objectStore.get()`方法用于读取数据，参数是主键的值。

遍历数据

遍历数据表格的所有记录，要使用指针对象 IDBCursor。

```javascript
function readAll() {
  var objectStore = db.transaction('person').objectStore('person');

  objectStore.openCursor().onsuccess = function (event) {
    var cursor = event.target.result;

    if (cursor) {
```

```
      console.log('Id: ' + cursor.key);
      console.log('Name: ' + cursor.value.name);
      console.log('Age: ' + cursor.value.age);
      console.log('Email: ' + cursor.value.email);
      cursor.continue();
    } else {
      console.log('没有更多数据了！ ');
    }
  };
}

readAll();
```

上面代码中，新建指针对象的`openCursor()`方法是一个异步操作，所以要监听`success`事件。

更新数据

更新数据要使用`IDBObject.put()`方法。

```javascript
function update() {
  var request = db.transaction(['person'], 'readwrite')
    .objectStore('person')
    .put({ id: 1, name: '李四', age: 35, email: 'lisi@example.com' });

  request.onsuccess = function (event) {
    console.log('数据更新成功');
  };

  request.onerror = function (event) {
    console.log('数据更新失败');
  }
}

update();
```

上面代码中，`put()`方法自动更新了主键为`1`的记录。

删除数据

`IDBObjectStore.delete()`方法用于删除记录。

```javascript
function remove() {
  var request = db.transaction(['person'], 'readwrite')
    .objectStore('person')
    .delete(1);
```

```
  request.onsuccess = function (event) {
    console.log('数据删除成功');
  };
}

remove();
```

使用索引

索引的意义在于，可以让你搜索任意字段，也就是说从任意字段拿到数据记录。如果不建立索引，默认只能搜索主键（即从主键取值）。

假定新建表格的时候，对`name`字段建立了索引。

```javascript
objectStore.createIndex('name', 'name', { unique: false });
```

现在，就可以从`name`找到对应的数据记录了。

```javascript
var transaction = db.transaction(['person'], 'readonly');
var store = transaction.objectStore('person');
var index = store.index('name');
var request = index.get('李四');

request.onsuccess = function (e) {
  var result = e.target.result;
  if (result) {
    // ...
  } else {
    // ...
  }
}
```

indexedDB 对象

浏览器原生提供`indexedDB`对象，作为开发者的操作接口。

indexedDB.open()

`indexedDB.open()`方法用于打开数据库。这是一个异步操作，但是会立刻返回一个IDBOpenDBRequest 对象。

```javascript
var openRequest = window.indexedDB.open('test', 1);
```

上面代码表示，打开一个名为`test`、版本为`1`的数据库。如果该数据库不存在，则会新建该数据库。

`open()`方法的第一个参数是数据库名称，格式为字符串，不可省略；第二个参数是数据库版本，是一个大于`0`的正整数（`0`将报错），如果该参数大于当前版本，会触发数据库升级。第二个参数可省略，如果数据库已存在，将打开当前版本的数据库；如果数据库不存在，将创建该版本的数据库，默认版本为`1`。

打开数据库是异步操作，通过各种事件通知客户端。下面是有可能触发的4种事件。

- **success**：打开成功。
- **error**：打开失败。
- **upgradeneeded**：第一次打开该数据库，或者数据库版本发生变化。
- **blocked**：上一次的数据库连接还未关闭。

第一次打开数据库时，会先触发`upgradeneeded`事件，然后触发`success`事件。

根据不同的需要，对上面4种事件监听函数。

```javascript
var openRequest = indexedDB.open('test', 1);
var db;

openRequest.onupgradeneeded = function (e) {
  console.log('Upgrading...');
}

openRequest.onsuccess = function (e) {
  console.log('Success!');
  db = openRequest.result;
}

openRequest.onerror = function (e) {
  console.log('Error');
  console.log(e);
}
```

上面代码有两个地方需要注意。首先，`open()`方法返回的是一个对象（IDBOpenDBRequest），监听函数就定义在这个对象上面。其次，`success`事件发生后，从`openRequest.result`属性可以拿到已经打开的`IndexedDB`数据库对象。

indexedDB.deleteDatabase()

`indexedDB.deleteDatabase()`方法用于删除一个数据库，参数为数据库的名字。它会立刻返回一个`IDBOpenDBRequest`对象，然后对数据库执行异步删除。删除操作的结果会通过事件通知，`IDBOpenDBRequest`对象可以监听以下事件。

- `success`：删除成功
- `error`：删除报错

```javascript
var DBDeleteRequest = window.indexedDB.deleteDatabase('demo');

DBDeleteRequest.onerror = function (event) {
  console.log('Error');
};

DBDeleteRequest.onsuccess = function (event) {
  console.log('success');
};
```

调用`deleteDatabase()`方法以后，当前数据库的其他已经打开的连接都会接收到`versionchange`事件。

注意，删除不存在的数据库并不会报错。

indexedDB.cmp()

`indexedDB.cmp()`方法比较两个值是否为 indexedDB 的相同的主键。它返回一个整数，表示比较的结果：`0`表示相同，`1`表示第一个主键大于第二个主键，`-1`表示第一个主键小于第二个主键。

```javascript
window.indexedDB.cmp(1, 2) // -1
```

注意，这个方法不能用来比较任意的 JavaScript 值。如果参数是布尔值或对象，它会报错。

```javascript
window.indexedDB.cmp(1, true) // 报错
window.indexedDB.cmp({}, {}) // 报错
```

IDBRequest 对象

IDBRequest 对象表示打开的数据库连接，`indexedDB.open()`方法和`indexedDB.deleteDatabase()`方法会返回这个对象。数据库的操作都是通过这个对象完成的。

这个对象的所有操作都是异步操作，要通过`readyState`属性判断是否完成，如果为`pending`就表示操作正在进行，如果为`done`就表示操作完成，可能成功也可能失败。

操作完成以后，触发`success`事件或`error`事件，这时可以通过`result`属性和`error`属性拿到操作结果。如果在`pending`阶段，就去读取这两个属性，是会报错的。

IDBRequest 对象有以下属性。

- `IDBRequest.readyState`：等于`pending`表示操作正在进行，等于`done`表示操作正在完成。
- `IDBRequest.result`：返回请求的结果。如果请求失败、结果不可用，读取该属性会报错。
- `IDBRequest.error`：请求失败时，返回错误对象。
- `IDBRequest.source`：返回请求的来源（比如索引对象或 ObjectStore）。
- `IDBRequest.transaction`：返回当前请求正在进行的事务，如果不包含事务，返回`null`。
- `IDBRequest.onsuccess`：指定`success`事件的监听函数。
- `IDBRequest.onerror`：指定`error`事件的监听函数。

IDBOpenDBRequest 对象继承了 IDBRequest 对象，提供了两个额外的事件监听属性。

- `IDBOpenDBRequest.onblocked`：指定`blocked`事件（`upgradeneeded`事件触发时，数据库仍然在使用）的监听函数。
- `IDBOpenDBRequest.onupgradeneeded`：`upgradeneeded`事件的监听函数。

IDBDatabase 对象

打开数据成功以后，可以从`IDBOpenDBRequest`对象的`result`属性上面，拿到一个`IDBDatabase`对象，它表示连接的数据库。后面对数据库的操作，都通过这个对象完成。

```javascript
var db;
var DBOpenRequest = window.indexedDB.open('demo', 1);

DBOpenRequest.onerror = function (event) {
  console.log('Error');
};

DBOpenRequest.onsuccess = function(event) {
  db = DBOpenRequest.result;
  // ...
};
```

属性

IDBDatabase 对象有以下属性。

- `IDBDatabase.name`：字符串，数据库名称。
- `IDBDatabase.version`：整数，数据库版本。数据库第一次创建时，该属性为空字符串。

- `IDBDatabase.objectStoreNames`：DOMStringList 对象（字符串的集合），包含当前数据的所有 object store 的名字。
- `IDBDatabase.onabort`：指定 abort 事件（事务中止）的监听函数。
- `IDBDatabase.onclose`：指定 close 事件（数据库意外关闭）的监听函数。
- `IDBDatabase.onerror`：指定 error 事件（访问数据库失败）的监听函数。
- `IDBDatabase.onversionchange`：数据库版本变化时触发（发生`upgradeneeded`事件，或调用`indexedDB.deleteDatabase()`）。

下面是`objectStoreNames`属性的例子。该属性返回一个 DOMStringList 对象，包含了当前数据库所有对象仓库的名称（即表名），可以使用 DOMStringList 对象的`contains`方法，检查数据库是否包含某个对象仓库。

```javascript
if (!db.objectStoreNames.contains('firstOS')) {
  db.createObjectStore('firstOS');
}
```

上面代码先判断某个对象仓库是否存在，如果不存在就创建该对象仓库。

方法

IDBDatabase 对象有以下方法。

- `IDBDatabase.close()`：关闭数据库连接，实际会等所有事务完成后再关闭。
- `IDBDatabase.createObjectStore()`：创建存放数据的对象仓库，类似于传统关系型数据库的表格，返回一个 IDBObjectStore 对象。该方法只能在`versionchange`事件监听函数中调用。
- `IDBDatabase.deleteObjectStore()`：删除指定的对象仓库。该方法只能在`versionchange`事件监听函数中调用。
- `IDBDatabase.transaction()`：返回一个 IDBTransaction 事务对象。

下面是`createObjectStore()`方法的例子。

```javascript
var request = window.indexedDB.open('demo', 2);

request.onupgradeneeded = function (event) {
  var db = event.target.result;

  db.onerror = function(event) {
    console.log('error');
  };

  var objectStore = db.createObjectStore('items');

  // ...
```

};
```

上面代码创建了一个名为`items`的对象仓库，如果该对象仓库已经存在，就会抛出一个错误。为了避免出错，需要用到下文的`objectStoreNames`属性，检查已有哪些对象仓库。

`createObjectStore()`方法还可以接受第二个对象参数，用来设置对象仓库的属性。

```javascript
db.createObjectStore('test', { keyPath: 'email' });
db.createObjectStore('test2', { autoIncrement: true });
```

上面代码中，`keyPath`属性表示主键（由于主键的值不能重复，所以上例存入之前，必须保证数据的`email`属性值都是不一样的），默认值为`null`；`autoIncrement`属性表示，是否使用自动递增的整数作为主键（第一个数据记录为1，第二个数据记录为2，以此类推），默认为`false`。一般来说，`keyPath`和`autoIncrement`属性只要使用一个就够了，如果两个同时使用，表示主键为递增的整数，且对象不得缺少`keyPath`指定的属性。

下面是`deleteObjectStore()`方法的例子。

```javascript
var dbName = 'sampleDB';
var dbVersion = 2;
var request = indexedDB.open(dbName, dbVersion);

request.onupgradeneeded = function(e) {
  var db = request.result;
  if (e.oldVersion < 1) {
    db.createObjectStore('store1');
  }

  if (e.oldVersion < 2) {
    db.deleteObjectStore('store1');
    db.createObjectStore('store2');
  }

  // ...
};
```

下面是`transaction()`方法的例子，该方法用于创建一个数据库事务，返回一个 IDBTransaction 对象。向数据库添加数据之前，必须先创建数据库事务。

```javascript
var t = db.transaction(['items'], 'readwrite');
```
```

`transaction()`方法接受两个参数：第一个参数是一个数组，里面是所涉及的对象仓库，通常是只有一个；第二个参数是一个表示操作类型的字符串。目前，操作类型只有两种：`readonly`（只读）和`readwrite`（读写）。添加数据使用`readwrite`，读取数据使用`readonly`。第二个参数是可选的，省略时默认为`readonly`模式。

IDBObjectStore 对象

IDBObjectStore 对象对应一个对象仓库（object store）。`IDBDatabase.createObjectStore()`方法返回的就是一个 IDBObjectStore 对象。

IDBDatabase 对象的`transaction()`返回一个事务对象，该对象的`objectStore()`方法返回 IDBObjectStore 对象，因此可以采用下面的链式写法。

```javascript
db.transaction(['test'], 'readonly')
  .objectStore('test')
  .get(X)
  .onsuccess = function (e) {}
```

属性

IDBObjectStore 对象有以下属性。

- `IDBObjectStore.indexNames`：返回一个类似数组的对象（DOMStringList），包含了当前对象仓库的所有索引。
- `IDBObjectStore.keyPath`：返回当前对象仓库的主键。
- `IDBObjectStore.name`：返回当前对象仓库的名称。
- `IDBObjectStore.transaction`：返回当前对象仓库所属的事务对象。
- `IDBObjectStore.autoIncrement`：布尔值，表示主键是否会自动递增。

方法

IDBObjectStore 对象有以下方法。

（1）IDBObjectStore.add()

`IDBObjectStore.add()`用于向对象仓库添加数据，返回一个 IDBRequest 对象。该方法只用于添加数据，如果主键相同会报错，因此更新数据必须使用`put()`方法。

```javascript
objectStore.add(value, key)
```

该方法接受两个参数，第一个参数是键值，第二个参数是主键，该参数可选，如果省略默认为`null`。

创建事务以后，就可以获取对象仓库，然后使用`add()`方法往里面添加数据了。

```javascript
var db;
var DBOpenRequest = window.indexedDB.open('demo', 1);

DBOpenRequest.onsuccess = function (event) {
  db = DBOpenRequest.result;
  var transaction = db.transaction(['items'], 'readwrite');

  transaction.oncomplete = function (event) {
    console.log('transaction success');
  };

  transaction.onerror = function (event) {
    console.log('transaction error: ' + transaction.error);
  };

  var objectStore = transaction.objectStore('items');
  var objectStoreRequest = objectStore.add({ foo: 1 });

  objectStoreRequest.onsuccess = function (event) {
    console.log('add data success');
  };

};
```

（2）IDBObjectStore.put()

`IDBObjectStore.put()`方法用于更新某个主键对应的数据记录，如果对应的键值不存在，则插入一条新的记录。该方法返回一个 IDBRequest 对象。

```javascript
objectStore.put(item, key)
```

该方法接受两个参数，第一个参数为新数据，第二个参数为主键，该参数可选，且只在自动递增时才有必要提供，因为那时主键不包含在数据值里面。

（3）IDBObjectStore.clear()

`IDBObjectStore.clear()`删除当前对象仓库的所有记录。该方法返回一个 IDBRequest 对象。

```javascript
objectStore.clear()
```

该方法不需要参数。

（4）IDBObjectStore.delete()

`IDBObjectStore.delete()`方法用于删除指定主键的记录。该方法返回一个 IDBRequest 对象。

```javascript
objectStore.delete(Key)
```

该方法的参数为主键的值。

（5）IDBObjectStore.count()

`IDBObjectStore.count()`方法用于计算记录的数量。该方法返回一个 IDBRequest 对象。

```javascript
IDBObjectStore.count(key)
```

不带参数时，该方法返回当前对象仓库的所有记录数量。如果主键或 IDBKeyRange 对象作为参数，则返回对应的记录数量。

（6）IDBObjectStore.getKey()

`IDBObjectStore.getKey()`用于获取主键。该方法返回一个 IDBRequest 对象。

```javascript
objectStore.getKey(key)
```

该方法的参数可以是主键值或 IDBKeyRange 对象。

（7）IDBObjectStore.get()

`IDBObjectStore.get()`用于获取主键对应的数据记录。该方法返回一个 IDBRequest 对象。

```javascript
objectStore.get(key)
```

（8）IDBObjectStore.getAll()

`DBObjectStore.getAll()`用于获取对象仓库的记录。该方法返回一个 IDBRequest 对象。

```javascript
// 获取所有记录
```

```
objectStore.getAll()

// 获取所有符合指定主键或 IDBKeyRange 的记录
objectStore.getAll(query)

// 指定获取记录的数量
objectStore.getAll(query, count)
```

（9）IDBObjectStore.getAllKeys()

`IDBObjectStore.getAllKeys()`用于获取所有符合条件的主键。该方法返回一个 IDBRequest 对象。

```javascript
// 获取所有记录的主键
objectStore.getAllKeys()

// 获取所有符合条件的主键
objectStore.getAllKeys(query)

// 指定获取主键的数量
objectStore.getAllKeys(query, count)
```

（10）IDBObjectStore.index()

`IDBObjectStore.index()`方法返回指定名称的索引对象 IDBIndex。

```javascript
objectStore.index(name)
```

有了索引以后，就可以针对索引所在的属性读取数据。

```javascript
var t = db.transaction(['people'], 'readonly');
var store = t.objectStore('people');
var index = store.index('name');

var request = index.get('foo');
```

上面代码打开对象仓库以后，先用`index()`方法指定获取`name`属性的索引，然后用`get()`方法读取某个`name`属性(`foo`)对应的数据。如果`name`属性不是对应唯一值，这时`get()`方法有可能取回多个数据对象。另外，`get()`是异步方法，读取成功以后，只能在`success`事件的监听函数中处理数据。

（11）IDBObjectStore.createIndex()

`IDBObjectStore.createIndex()`方法用于新建当前数据库的一个索引。该方法只能在`VersionChange`监听函数里面调用。

```javascript
objectStore.createIndex(indexName, keyPath, objectParameters)
```

该方法可以接受三个参数。

- indexName：索引名
- keyPath：主键
- objectParameters：配置对象（可选）

第三个参数可以配置以下属性。

- unique：如果设为`true`，将不允许重复的值
- multiEntry：如果设为`true`，对于有多个值的主键数组，每个值将在索引里面新建一个条目，否则主键数组对应一个条目。

假定对象仓库中的数据记录都是如下的`person`类型。

```javascript
var person = {
  name: name,
  email: email,
  created: new Date()
};
```

可以指定这个对象的某个属性来建立索引。

```javascript
var store = db.createObjectStore('people', { autoIncrement: true });

store.createIndex('name', 'name', { unique: false });
store.createIndex('email', 'email', { unique: true });
```

上面代码告诉索引对象，`name`属性不是唯一值，`email`属性是唯一值。

（12）IDBObjectStore.deleteIndex()

`IDBObjectStore.deleteIndex()`方法用于删除指定的索引。该方法只能在`VersionChange`监听函数里面调用。

```javascript
```

```
objectStore.deleteIndex(indexName)
```

（13）IDBObjectStore.openCursor()

`IDBObjectStore.openCursor()`用于获取一个指针对象。

```javascript
IDBObjectStore.openCursor()
```

指针对象可以用来遍历数据。该对象也是异步的，有自己的`success`和`error`事件，可以对它们指定监听函数。

```javascript
var t = db.transaction(['test'], 'readonly');
var store = t.objectStore('test');

var cursor = store.openCursor();

cursor.onsuccess = function (event) {
  var res = event.target.result;
  if (res) {
    console.log('Key', res.key);
    console.dir('Data', res.value);
    res.continue();
  }
}
```

监听函数接受一个事件对象作为参数，该对象的`target.result`属性指向当前数据记录。该记录的`key`和`value`分别返回主键和键值（即实际存入的数据）。`continue()`方法将光标移到下一个数据对象，如果当前数据对象已经是最后一个数据了，则光标指向`null`。

`openCursor()`方法的第一个参数是主键值，或者一个 IDBKeyRange 对象。如果指定该参数，将只处理包含指定主键的记录；如果省略，将处理所有的记录。该方法还可以接受第二个参数，表示遍历方向，默认值为`next`，其他可能的值为`prev`、`nextunique`和`prevunique`。后两个值表示如果遇到重复值，会自动跳过。

（14）IDBObjectStore.openKeyCursor()

`IDBObjectStore.openKeyCursor()`用于获取一个主键指针对象。

```javascript
IDBObjectStore.openKeyCursor()
```

IDBTransaction 对象

IDBTransaction 对象用来异步操作数据库事务，所有的读写操作都要通过这个对象进行。

`IDBDatabase.transaction()`方法返回的就是一个 IDBTransaction 对象。

```javascript
var db;
var DBOpenRequest = window.indexedDB.open('demo', 1);

DBOpenRequest.onsuccess = function(event) {
  db = DBOpenRequest.result;
  var transaction = db.transaction(['demo'], 'readwrite');

  transaction.oncomplete = function (event) {
    console.log('transaction success');
  };

  transaction.onerror = function (event) {
    console.log('transaction error: ' + transaction.error);
  };

  var objectStore = transaction.objectStore('demo');
  var objectStoreRequest = objectStore.add({ foo: 1 });

  objectStoreRequest.onsuccess = function (event) {
    console.log('add data success');
  };

};
```

事务的执行顺序是按照创建的顺序，而不是发出请求的顺序。

```javascript
var trans1 = db.transaction('foo', 'readwrite');
var trans2 = db.transaction('foo', 'readwrite');
var objectStore2 = trans2.objectStore('foo')
var objectStore1 = trans1.objectStore('foo')
objectStore2.put('2', 'key');
objectStore1.put('1', 'key');
```

上面代码中，`key`对应的键值最终是`2`，而不是`1`。因为事务`trans1`先于`trans2`创建，所以首先执行。

注意，事务有可能失败，只有监听到事务的`complete`事件，才能保证事务操作成功。

IDBTransaction 对象有以下属性。

- `IDBTransaction.db`：返回当前事务所在的数据库对象 IDBDatabase。

- `IDBTransaction.error`：返回当前事务的错误。如果事务没有结束，或者事务成功结束，或者被手动终止，该方法返回`null`。
- `IDBTransaction.mode`：返回当前事务的模式，默认是`readonly`（只读），另一个值是`readwrite`。
- `IDBTransaction.objectStoreNames`：返回一个类似数组的对象 DOMStringList，成员是当前事务涉及的对象仓库的名字。
- `IDBTransaction.onabort`：指定`abort`事件（事务中断）的监听函数。
- `IDBTransaction.oncomplete`：指定`complete`事件（事务成功）的监听函数。
- `IDBTransaction.onerror`：指定`error`事件（事务失败）的监听函数。

IDBTransaction 对象有以下方法。

- `IDBTransaction.abort()`：终止当前事务，回滚所有已经进行的变更。
- `IDBTransaction.objectStore(name)`：返回指定名称的对象仓库 IDBObjectStore。

IDBIndex 对象

IDBIndex 对象代表数据库的索引，通过这个对象可以获取数据库里面的记录。数据记录的主键默认就是带有索引，IDBIndex 对象主要用于通过除主键以外的其他键，建立索引获取对象。

IDBIndex 是持久性的键值对存储。只要插入、更新或删除数据记录，引用的对象库中的记录，索引就会自动更新。

`IDBObjectStore.index()`方法可以获取 IDBIndex 对象。

```javascript
var transaction = db.transaction(['contactsList'], 'readonly');
var objectStore = transaction.objectStore('contactsList');
var myIndex = objectStore.index('lName');

myIndex.openCursor().onsuccess = function (event) {
  var cursor = event.target.result;
  if (cursor) {
    var tableRow = document.createElement('tr');
    tableRow.innerHTML =   '<td>' + cursor.value.id + '</td>'
                         + '<td>' + cursor.value.lName + '</td>'
                         + '<td>' + cursor.value.fName + '</td>'
                         + '<td>' + cursor.value.jTitle + '</td>'
                         + '<td>' + cursor.value.company + '</td>'
                         + '<td>' + cursor.value.eMail + '</td>'
                         + '<td>' + cursor.value.phone + '</td>'
                         + '<td>' + cursor.value.age + '</td>';
    tableEntry.appendChild(tableRow);

    cursor.continue();
  } else {
    console.log('Entries all displayed.');
```

```
  }
};
```

IDBIndex 对象有以下属性。

- `IDBIndex.name`：字符串，索引的名称。
- `IDBIndex.objectStore`：索引所在的对象仓库。
- `IDBIndex.keyPath`：索引的主键。
- `IDBIndex.multiEntry`：布尔值，针对`keyPath`为数组的情况，如果设为`true`，创建数组时，每个数组成员都会有一个条目，否则每个数组都只有一个条目。
- `IDBIndex.unique`：布尔值，表示创建索引时是否允许相同的主键。

IDBIndex 对象有以下方法，它们都是异步的，立即返回的都是一个 IDBRequest 对象。

- `IDBIndex.count()`：用来获取记录的数量。它可以接受主键或 IDBKeyRange 对象作为参数，这时只返回符合主键的记录数量，否则返回所有记录的数量。
- `IDBIndex.get(key)`：用来获取符合指定主键的数据记录。
- `IDBIndex.getKey(key)`：用来获取指定的主键。
- `IDBIndex.getAll()`：用来获取所有的数据记录。它可以接受两个参数，都是可选的，第一个参数用来指定主键，第二个参数用来指定返回记录的数量。如果省略这两个参数，则返回所有记录。由于获取成功时，浏览器必须生成所有对象，所以对性能有影响。如果数据集比较大，建议使用 IDBCursor 对象。
- `IDBIndex.getAllKeys()`：该方法与`IDBIndex.getAll()`方法相似，区别是获取所有主键。
- `IDBIndex.openCursor()`：用来获取一个 IDBCursor 对象，用来遍历索引里面的所有条目。
- `IDBIndex.openKeyCursor()`：该方法与`IDBIndex.openCursor()`方法相似，区别是遍历所有条目的主键。

IDBCursor 对象

IDBCursor 对象代表指针对象，用来遍历数据仓库（IDBObjectStore）或索引（IDBIndex）的记录。

IDBCursor 对象一般通过`IDBObjectStore.openCursor()`方法获得。

```javascript
var transaction = db.transaction(['rushAlbumList'], 'readonly');
var objectStore = transaction.objectStore('rushAlbumList');

objectStore.openCursor(null, 'next').onsuccess = function(event) {
  var cursor = event.target.result;
  if (cursor) {
    var listItem = document.createElement('li');
      listItem.innerHTML = cursor.value.albumTitle + ', ' + cursor.value.year;
      list.appendChild(listItem);
```

```
      console.log(cursor.source);
      cursor.continue();
    } else {
      console.log('Entries all displayed.');
    }
  };
};
```

IDBCursor 对象的属性。

- `IDBCursor.source`：返回正在遍历的对象仓库或索引。
- `IDBCursor.direction`：字符串，表示指针遍历的方向。共有四个可能的值：next（从头开始向后遍历）、nextunique（从头开始向后遍历，重复的值只遍历一次）、prev（从尾部开始向前遍历）、prevunique（从尾部开始向前遍历，重复的值只遍历一次）。该属性通过`IDBObjectStore.openCursor()`方法的第二个参数指定，一旦指定就不能改变了。
- `IDBCursor.key`：返回当前记录的主键。
- `IDBCursor.value`：返回当前记录的数据值。
- IDBCursor.primaryKey：返回当前记录的主键。对于数据仓库（objectStore）来说，这个属性等同于 IDBCursor.key；对于索引，IDBCursor.key 返回索引的位置值，该属性返回数据记录的主键。

IDBCursor 对象有如下方法。

- `IDBCursor.advance(n)`：指针向前移动 n 个位置。
- `IDBCursor.continue()`：指针向前移动一个位置。它可以接受一个主键作为参数，这时会跳转到这个主键。
- `IDBCursor.continuePrimaryKey()`：该方法需要两个参数，第一个是`key`，第二个是`primaryKey`，将指针移到符合这两个参数的位置。
- `IDBCursor.delete()`：用来删除当前位置的记录，返回一个 IDBRequest 对象。该方法不会改变指针的位置。
- `IDBCursor.update()`：用来更新当前位置的记录，返回一个 IDBRequest 对象。它的参数是要写入数据库的新的值。

IDBKeyRange 对象

IDBKeyRange 对象代表数据仓库（object store）里面的一组主键。根据这组主键，可以获取数据仓库或索引里面的一组记录。

IDBKeyRange 可以只包含一个值，也可以指定上限和下限。它有四个静态方法，用来指定主键的范围。

- `IDBKeyRange.lowerBound()`：指定下限。

- `IDBKeyRange.upperBound()`：指定上限。
- `IDBKeyRange.bound()`：同时指定上下限。
- `IDBKeyRange.only()`：指定只包含一个值。

下面是一些代码实例。

```javascript
// All keys ≤ x
var r1 = IDBKeyRange.upperBound(x);

// All keys < x
var r2 = IDBKeyRange.upperBound(x, true);

// All keys ≥ y
var r3 = IDBKeyRange.lowerBound(y);

// All keys > y
var r4 = IDBKeyRange.lowerBound(y, true);

// All keys ≥ x && ≤ y
var r5 = IDBKeyRange.bound(x, y);

// All keys > x &&< y
var r6 = IDBKeyRange.bound(x, y, true, true);

// All keys > x && ≤ y
var r7 = IDBKeyRange.bound(x, y, true, false);

// All keys ≥ x &&< y
var r8 = IDBKeyRange.bound(x, y, false, true);

// The key = z
var r9 = IDBKeyRange.only(z);
```

`IDBKeyRange.lowerBound()`、`IDBKeyRange.upperBound()`、`IDBKeyRange.bound()`这三个方法默认包括端点值，可以传入一个布尔值，修改这个属性。

与之对应，IDBKeyRange 对象有四个只读属性。

- `IDBKeyRange.lower`：返回下限
- `IDBKeyRange.lowerOpen`：布尔值，表示下限是否为开区间（即下限是否排除在范围之外）
- `IDBKeyRange.upper`：返回上限
- `IDBKeyRange.upperOpen`：布尔值，表示上限是否为开区间（即上限是否排除在范围之外）

IDBKeyRange 实例对象生成以后，将它作为参数输入 IDBObjectStore 或 IDBIndex 对象的`openCursor()`方法，就可以在所设定的范围内读取数据。

```javascript
```

```
var t = db.transaction(['people'], 'readonly');
var store = t.objectStore('people');
var index = store.index('name');

var range = IDBKeyRange.bound('B', 'D');

index.openCursor(range).onsuccess = function (e) {
  var cursor = e.target.result;
  if (cursor) {
    console.log(cursor.key + ':');

    for (var field in cursor.value) {
      console.log(cursor.value[field]);
    }
    cursor.continue();
  }
}
```

IDBKeyRange 有一个实例方法`includes(key)`，返回一个布尔值，表示某个主键是否包含在当前这个主键组之内。

```javascript
var keyRangeValue = IDBKeyRange.bound('A', 'K', false, false);

keyRangeValue.includes('F') // true
keyRangeValue.includes('W') // false
```

参考链接

- Raymond Camden, [Working With IndexedDB – Part 1](http://net.tutsplus.com/tutorials/javascript-ajax/working-with-indexeddb/)
- Raymond Camden, [Working With IndexedDB – Part 2](http://net.tutsplus.com/tutorials/javascript-ajax/working-with-indexeddb-part-2/)
- Raymond Camden, [Working With IndexedDB - Part 3](https://code.tutsplus.com/tutorials/working-with-indexeddb-part-3--net-36220)
- Tiffany Brown, [An Introduction to IndexedDB](http://dev.opera.com/articles/introduction-to-indexeddb/)
- David Fahlander, [Breaking the Borders of IndexedDB](https://hacks.mozilla.org/2014/06/breaking-the-borders-of-indexeddb/)
- TutorialsPoint, [HTML5 - IndexedDB](https://www.tutorialspoint.com/html5/html5_indexeddb.htm)