使用IndexedDB存储数据

# IndexedDB介绍

Indexed Database API，或者简称为 IndexedDB，是在浏览器中保存结构化数据的一种数据库。IndexedDB 的思想是创建一套 API，方便保存和读取 JavaScript 对象，同时还支持查询及搜索。

IndexedDB 最大的特色是使用对象保存数据，而不是使用表来保存数据。

可以存储包括字符串、对象、文件等在内的数据类型。

IndexedDB 设计的操作完全是异步进行的。大多数操作会以请求方式进行，但这些操作会在后期执行，然后如果成功则返回结果，如果失败则返回错误。

使用IndexedDB的一般流程：

1. 打开一个database。
2. 在打开的database中创建一个objectStore。
3. 通过一个transaction来对database进行操作（常见的增、删、改、查操作）。
4. 等待操作完成的回调。

# 打开database

打开一个IndexedDB只需要使用下面的代码就可以完成：

var request = indexedDB.open(name, version);

上述代码中，name为数据库的名称，version为数据库版本号，要求为非负整数，可以不传入。如果尝试传入负数则会引发Value is outside the 'unsigned long long' value range. 异常，传入版本号比当前数据库版本号低的整数会触发error事件，传入小数则会自动转换成整数。

传入的version比当前的version高的时候会触发upgradeneeded事件。某些特定操作（创建和删除objectStore、创建和删除索引）只能在此事件触发的时候进行。

完整的测试代码：

var r = indexedDB.open("TEST");

//添加事件监听

r.onupgradeneeded = function (e) {

console.log(e.type+"---"+e.target.result.version);

};

r.onsuccess = function (e) {

console.log(e.type+"---"+e.target.result.version);

e.target.result.close();

};

r.onerror = function (e) {

console.log(e.type+"---"+e.target.error.message);

};

//升级版本号

r = indexedDB.open("TEST",3);

r.onupgradeneeded = function (e) {

console.log(e.type+"---"+e.target.result.version);

};

r.onsuccess = function (e) {

console.log(e.type+"---"+e.target.result.version);

e.target.result.close();

};

r.onerror = function (e) {

console.log(e.type+"---"+e.target.error.message);

};

//直接抛出异常

try{

r = indexedDB.open("TEST",-1);

}catch(e){

console.log(e.message);

}

//触发error事件

r = indexedDB.open("TEST",1);

r.onupgradeneeded = function (e) {

console.log(e.type+"---"+e.target.result.version);

};

r.onsuccess = function (e) {

console.log(e.type+"---"+e.target.result.version);

e.target.result.close();

};

r.onerror = function (e) {

console.log(e.type+"---"+e.target.error.message);

};

# 创建objectStore

打开一个新的DB之后首先需要创建一个objecStore（类似于关系型数据库的表，下面统一用os指代）。创建和修改os的操作必须在upgradeneeded事件触发的时候进行，否则会报异常。

创建和修改os基本语法如下：

var db = event.target.result;

var store = db.createObjectStore(objectStoreName, {

keyPath: keyPath,

autoIncrement: false

});

//判断需要创建的objectStore是否已经存在

if (db.objectStoreNames.contains(objectStoreName)) {

console.log(objectStoreName+”已经存在”);

//删除objectStore对象

db.deleteObjectStore(objectStoreName);

}

createObjectStore 方法接受两个参数，objectStoreName是要创建的os对象的名称，第二个参数是一个对象，keyPath是要os要使用的key名，可以用来区别os中不同的对象（可以理解成传统数据库中的主键），autoIncrement表示该值是否要自增长。

deleteObjectStore方法接受一个参数，objectStoreName是要删除的os对象的名称。

现在一个os就创建好了，可以通过keyPath去查找os中的值（查找操作在后面提及）。

如果还想再os中创建其他可以，就可以通过createIndex方法。

os.createIndex(indexName,keyName, optional);

indexName表示需要创建的索引的别名，keyName是存储在os中的对象的具体key名，optional是一个对象，具体可能的参数如下：

属性 描述

unique 表示该index是否唯一。

multiEntry 表示该key的value是数组的情况下在该索引下是否按照多条记录显示。

如果想删除索引可以调用deleteIndex方法。

os. deleteIndex (indexNaml);

关于os和index操作的完整测试代码：

r.onupgradeneeded = function (e) {

var db=e.target.result;

//判断是否存在

if(db.objectStoreNames.contains("User")){

return;

}

//创建os

var os=db.createObjectStore("User",{

keyPath:"id",

autoIncrement:false

});

//创建索引

os.createIndex("by\_name","name",{unique:false});

os.createIndex("address","address",{unique:true});

os.createIndex("age","age",{unique:false});

try{

//创建key值相同的所以将会抛出异常

os.createIndex("age","age",{unique:false});

}catch(e){

console.log(e.message);

}

//删除索引

os.deleteIndex("age");

//删除空的索引将会抛出异常

try{

os.deleteIndex("age");

}catch(e){

console.log(e.message);

}

//删除创建的os

db.deleteObjectStore("User");

try{

//在空的os上创建索引会抛出异常

os.createIndex("age","age",{unique:false});

}catch(e){

console.log(e.message);

}

};

# 操作数据

在indexedDB中对os的操作需要通过一个事务（transaction）来进行。

首先来了解下事务的概念：

IDBDataBase下面的transaction方法会返回一个IDBTransaction对象。IDBTransaction事件机制提供了一个异步的操作IDB的方式。所有的读、写操作都是在事务中完成。在事务进行的过程中如果中间某个操作触发异常，会导致之前所有的操作都被回滚，继而触发error和abort事件。也可以手动调用tx.abort方法，去触发abort事件。如果所有操作都完成则会触发complete方法。

var tx = db.transaction(<Array|String>osNames, access)

osNames可以是数组或者字符串，表示需要进行操作的os的名称。

access表示事务需要以哪种方式（”readonly”,”readwrite”）去打开这个tx。

通过tx来获取想要操作的os。

var os = tx.objectStore(osName);

之后进行增、删、改、查的操作。

os.add(value);

os.delete(key);

os.put(value);

os.get(key);

add方法向os添加一条新的数据，如果数据已经存在则会触发error事件。

delete方法通过传入要删除的数据的key值，来删除一条数据。删除不存在的数据仍旧是触发success事件。

put方法也是向os中添加一条新的数据，不同add的是如果数据存在则会覆盖之前的数据。

get方法用来获取一条数据。获取不存在的数据不会触发异常，但是e.target.result的值为空。

测试代码：

r.onupgradeneeded = function(e) {

var db = e.target.result;

var os = db.createObjectStore("User", {

keyPath: "address",

autoIncrement: false

});

//创建索引

os.createIndex("by\_name", "name", {

unique: false

});

os.createIndex("age", "age", {

unique: false

});

};

r.onsuccess = function(e) {

var db = e.target.result;

var users = [{

name: "Bruce",

address: "No.2,ST",

age: "22"

}, {

name: "Bruce",

address: "No.1,ST",

age: "22"

}, {

name: "Lee",

address: "No.2,LA",

age: "18"

}];

var tx = db.transaction(["User"], "readwrite");

var os = tx.objectStore("User");

var r = [];

tx.onabort = function(e) {

console.log(e.type + "---" + e.target.db.name);

}

tx.onerror = function(e) {

console.log(e.type + "---" + e.target.error.message);

}

tx.oncomplete = function(e) {

console.log(e.type + "---" + e.target.db.name);

}

for (var i = users.length - 1; i >= 0; i--) {

try {

//增加数据

r[0] = os.add(users[i]);

} catch (e) {

console.log(e.message);

}

}

os.put(users[1]);

os.get(“No.1,ST”);

os.delete(“No.1,ST”);

};

# 使用索引(Index)来访问数据

除了通过keyPath来访问数据之外，还可以通过索引（index）来访问数据。

通过os. index方法可以获取之前创建的index，os.indexNames属性会返回之前参见的所有index的name。

var i = os.index(indexName);

var iNames=os.indexNames;

i.get(key);

i.getAll();//只有火狐支持

i.getAllKeys();//只有火狐支持

i.getKey(key);//返回keyPath的值

# 使用游标（cursor）获取数据

os和index对象下提供的get方法只能获取你所知道的key的那条数据，如果需要遍历os或者index下的数据，需要用到Cursor。

os.openCursor();

i.openCursor();

os.openKeyCursor();

i.openKeyCursor();

openCursor方法和openKeyCursor可以接受一系列参数用来限制想要从objectStore中取的值的范围和取值方向（倒序或者正序）。

参数：

range 取值范围，默认取全部值，详细的可以参见下表。

direction “prev”或者”next”,默认是”next”。

| Range | Code |
| --- | --- |
| All keys ≤ **x** | [IDBKeyRange.upperBound](http://devdocs.io/dom/idbkeyrange/upperbound)(x) |
| All keys < **x** | [IDBKeyRange.upperBound](http://devdocs.io/dom/idbkeyrange/upperbound)(x, true) |
| All keys ≥ **y** | [IDBKeyRange.lowerBound](http://devdocs.io/dom/idbkeyrange/lowerbound)(y) |
| All keys > **y** | [IDBKeyRange.lowerBound](http://devdocs.io/dom/idbkeyrange/lowerbound)(y, true) |
| All keys ≥ **x** && ≤ **y** | [IDBKeyRange.bound](http://devdocs.io/dom/idbkeyrange/bound)(x, y) |
| All keys > **x** &&< **y** | [IDBKeyRange.bound](http://devdocs.io/dom/idbkeyrange/bound)(x, y, true, true) |
| All keys > **x** && ≤ **y** | [IDBKeyRange.bound](http://devdocs.io/dom/idbkeyrange/bound)(x, y, true, false) |
| All keys ≥ **x** &&< **y** | [IDBKeyRange.bound](http://devdocs.io/dom/idbkeyrange/bound)(x, y, false, true) |
| The key = **z** | [IDBKeyRange.only](http://devdocs.io/dom/idbkeyrange/only)(z) |

使用cursor查询的一个例子：

var cursorRequest=os.openCursor();

cursorRequest.onsuccess=function(e){

var cursor = event.target.result;

if(cursor){

console.log(cursor.value.name);

cursor.continue();

}else{

db.close();//结束查询

}

}

上面列举了使用indexedDB大致的流程和主要的api，更多详细的资料可以参考文档。附件是一个完整的使用indexedDB的例子，提供大家作为参考。

