IndexedDB相关笔记

目 录

1. IndexedDB数据库概述
   1. 数据库概述
      1. 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。

* + 1. IndexedDB的特点
       1. 键值对储存（Key-Value）

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。在对象仓库中，数据以“键值对”的形式保存，每一个数据都有对应的键名，且键名必须是独一无二的，不能有重复，否则会抛出错误。

* + - 1. 异步API（asynchronous API）

IndexedDB数据库在执行增、删、改和查的操作时不会锁死浏览器，用户依然可以进行其它操作。相比之下，localStorage的操作都是同步的。异步设计是为了防止大量数据的读写时拖慢网页，而影响用户的网站体验。

* + - 1. 支持事务（transaction）

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，数据库回到事务发生之前的状态，不存在只改写一部分数据的情况。

* + - 1. 同域限制

IndexedDB也受到同域限制，每一个数据库对应创建该数据库的域名。来自不同域名的网页，只能访问自身域名下的数据库，而不能访问其他域名下的数据库。

* + - 1. 存储空间大

IndexedDB的存储空间比localStorage大得多，一般来说不少于250MB。IE的储存上限是250MB，Chrome和Opera是硬盘剩余空间的某个百分比，Firefox则没有上限。

* + - 1. 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。

* 1. 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。

1. 数据库：IDBDatabase 对象
2. 对象仓库：IDBObjectStore 对象
3. 索引： IDBIndex 对象
4. 事务： IDBTransaction 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。

* + 1. 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。

* + 1. 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。

{ 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。

* + 1. 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。

* + 1. 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。

* 1. 与别的前端存储方式相对比
     1. IndexedDB VS LocalStorage

IndexedDB和LocalStorage都是用来在浏览器里存储数据，但它们使用不同的技术，有不同的用途，你需要根据自己的情况适当的选择使用哪种。

LocalStorage是用key-value键值模式存储数据，但跟IndexedDB不一样的是，它的数据并不是按对象形式存储。它存储的数据都是字符串形式。如果你想让LocalStorage存储对象，你需要借助JSON.stringify()能将对象变成字符串形式，再用JSON.parse()将字符串还原成对象。但如果要存储大量的复杂的数据，这并不是一种很好的方案。毕竟，localstorage就是专门为小数量数据设计的，它的api是同步的。

IndexedDB很适合存储大量数据，它的API是异步调用的。IndexedDB使用索引存储数据，各种数据库操作放在事务中执行。IndexedDB甚至还支持简单的数据类型。IndexedDB比localstorage强大得多，但它的API也相对复杂。对于简单的数据，你应该继续使用localstorage，但当你希望存储大量数据时，IndexedDB会明显的更适合，IndexedDB能提供给你更为复杂的查询数据的方式。

* + 1. IndexedDB VS Web SQL

WebSQL也是一种在浏览器里存储数据的技术，跟IndexedDB不同的是，IndexedDB更像是一个NoSQL数据库，而WebSQL更像是关系型数据库，使用SQL查询数据。W3C已经不再支持这种技术。具体情况请看：http://www.w3.org/TR/webdatabase/。

**因为不再支持，所以你就不要在项目中使用这种技术了。**

* + 1. IndexedDB VS Cookies

Cookies(小甜点)听起来很好吃，但实际上并不是。每次HTTP接受和发送都会传递Cookies数据，它会占用额外的流量。例如，如果你有一个10KB的Cookies数据，发送10次请求，那么，总计就会有100KB的数据在网络上传输。Cookies只能是字符串。浏览器里存储Cookies的空间有限，很多用户禁止浏览器使用Cookies。所以，Cookies只能用来存储小量的非关键的数据。

* 1. IndexedDB架构及技术实现

IndexedDB的架构很像在一些流行的服务器端NoSQL数据库实现中的设计典范类型。面向对象数据通过object stores(对象仓库)进行持久化，所有操作基于请求同时在事务范围内执行；事件生命周期使你能够控制数据库的配置，错误通过错误冒泡来使用API管理。

* + 1. IndexedDB之打开数据库的请求生命周期（Life cycle）

浏览器原生提供indexedDB对象，可以通过**window.indexedDB**来直接获取到浏览器提供的该对象，作为开发者的操作接口。**IndexedDB.open**方法用于打开浏览器本地数据库。

IndexedDB使用事件生命周期管理数据库的打开和配置操作。

* + 1. IndexedDB之基于请求（Request）

对数据库的每次操作，可以描述为通过一个请求打开数据库，访问一个object store，再继续。IndexedDB API天生是基于请求的，这也是异步API本性所示。对于在数据库执行的每次操作，都必须首先为这个操作创建一个请求。当该请求完成，可以响应由请求结果产生的事件或错误。

* + 1. IndexedDB之对象仓库（Object Store）

Object store是IndexedDB数据库的基础。Object store相当于关系型数据库中的一张张记录数据的表。Object stores中包括一个或多个索引(index)，在store中按照一对键-值操作，这提供了一种快速定位数据的方法。

* + 1. IndexedDB之一切操作基于事务（Transaction）

不同于一些传统的关系数据库的实现，每一个对数据库操作是在一个事务的上下文中执行的。事务范围一次影响一个或多个object stores，你通过传入一个object store名字的数组到创建事务范围的函数来定义。

创建事务的第二个参数是事务模式。当请求一个事务时，必须决定是按照只读(ReadOnly)还是读写(ReadWrite)模式请求访问。事务是资源密集型的，所以如果你不需要更改data store中的数据，你只需要以只读模式对object stores集合进行请求访问。

* + 1. IndexedDB之错误冒泡

当然，有时候，请求可能不会按预期完成。

IndexedDB API通过错误冒泡功能来帮助跟踪和管理错误。如果一个特定的请求遇到错误，你可以尝试在请求对象上处理错误，或者你可以允许错误通过调用栈冒泡向上传递。这个冒泡天性，使得你不需要为每个请求实现特定错误处理操作，而是可以选择只在一个更高级别上添加错误处理，它给你一个机会，保持你的错误处理代码简洁。

Try……catch机制。

* 1. w3c中关于IndexedDB基本用法及API

http://www.w3.org/TR/IndexedDB/#dfn-invalidstateerror

* 1. IndexedDB参考资料

IndexedDB:浏览器里内置的数据库， Web骇客

http://www.webhek.com/indexeddb/

前端的数据库：IndexedDB入门（很全面）

http://web.jobbole.com/81793/

IndexedDB:浏览器端数据库， 阮一峰《JavaScript标准参考教程(alpha)》

http://javascript.ruanyifeng.com/bom/indexeddb.html

HTML5本地存储——IndexedDB

http://www.cnblogs.com/dolphinX/p/3415761.html (一、基本使用)

http://www.cnblogs.com/dolphinX/p/3415761.html (二、索引)

HTML5之IndexedDB使用详解

http://rainsilence.iteye.com/blog/984569

W3C: Indexed Database API

http://www.w3.org/TR/IndexedDB/#dfn-invalidstateerror

Dive into HTML5: 本地存储

http://www.devbean.net/2011/06/dive-into-html5-localstorage/

Angular-indexedDB

https://github.com/bramski/angular-indexedDB

HTML5 Local Storage本地存储

http://www.cnblogs.com/xiaowei0705/archive/2011/04/19/2021372.html

* 1. 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。

* + 1. 打开数据库

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

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。

request.onerror = function (event) {

console.log('数据库打开报错');

};

* + - 1. success 事件

success事件表示成功打开数据库。

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。

* + - 1. 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。

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。

* + 1. 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）。

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。

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。

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。

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）。

* + 1. 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。

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。

* + 1. 读取数据

读取数据也是通过事务完成。

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。

* + 1. 遍历数据

遍历数据表格的所有记录，要使用指针对象 IDBCursor。

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。

* + 1. 更新数据

更新数据要使用IDBObject.put()方法。

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。

* + 1. 删除数据

IDBObjectStore.delete()方法用于删除记录。

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();

* + 1. 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。

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

现在，就可以从name找到对应的数据记录了。

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 {

// ...

}

}