随着前端越来越重,已经不是当年的一个页面配几个第三方js库实现一些花俏的效果,而是出现了各种各样的框架解决方案.

当我们项目实现前后端分离时,甚至一些逻辑处理也会放到前端来处理.

所以随随便便打开一个页面上百k应该也是正常,甚至更大的页面在考验的前端.

但是伴随成长的后端,尤其数存储方面貌似发展缓慢,总有跟不上的赶脚!

想当年我们使用关系型数据库,当达到一个瓶颈时,解决起来是不计后果的,读写分离,拆表拆库,做service中间件,然后给中间件负载均衡等等来减轻前端并发造成的后端数据压力.

如今的mongodb 做一个分布式貌似轻轻松松, redis 做一层缓存让你有飞一样的感觉.我们终于谈到了缓存.

缓存应用场景很多,空间换时间的原理,会占用一点点内存,但是给你的web服务提升明显的性能.

并不是所有的数据都要放到缓存.不长更新的,没有实时性的数据可以暂时用缓存来处理.比如我的博客首页大概一天更新一次,变化并不是很大.

就可以把首页数据放到缓存来处理.

虽然我的数据库用的LevelDB ,号称也是处理海量数据的本地数据库,关于高性能LevelDB使用,请点击查看另外一篇博文

就算性能再高也是走IO,毕竟不如直接命中内存来的简单粗暴.

需求:

我只缓存首页,所以一定要轻量级,默认缓存一天,当发布新博客时,清楚缓存即可.

解决方案:

node-cache

官方地址:https://github.com/ptarjan/node-cache

绝对够小巧,还满足我简单的需求.

node-cache API 使用请移步到github 查看.

源码如下:

'use strict';

var cache = Object.create(null);

var debug = false;

var hitCount = 0;

var missCount = 0;

var size = 0;

exports.put = function(key, value, time, timeoutCallback) {

if (debug) {

console.log('caching: %s = %j (@%s)', key, value, time);

}

var oldRecord = cache[key];

if (oldRecord) {

clearTimeout(oldRecord.timeout);

} else {

size++;

}

var expire = time + Date.now();

var record = {

value: value,

expire: expire

};

if (!isNaN(expire)) {

var timeout = setTimeout(function() {

exports.del(key);

if (typeof timeoutCallback === 'function') {

timeoutCallback(key);

}

}, time);

record.timeout = timeout;

}

cache[key] = record;

};

exports.del = function(key) {

var canDelete = true;

var oldRecord = cache[key];

if (oldRecord) {

clearTimeout(oldRecord.timeout);

if (!isNaN(oldRecord.expire) && oldRecord.expire < Date.now()) {

canDelete = false;

}

} else {

canDelete = false;

}

if (canDelete) {

size--;

delete cache[key];

}

return canDelete;

};

exports.clear = function() {

for (var key in cache) {

var oldRecord = cache[key];

if (oldRecord) {

clearTimeout(oldRecord.timeout);

}

}

size = 0;

cache = Object.create(null);

if (debug) {

hitCount = 0;

missCount = 0;

}

};

exports.get = function(key) {

var data = cache[key];

if (typeof data != "undefined") {

if (isNaN(data.expire) || data.expire >= Date.now()) {

if (debug) hitCount++;

return data.value;

} else {

// free some space

if (debug) missCount++;

size--;

delete cache[key];

}

} else if (debug) {

missCount++;

}

return null;

};

exports.size = function() {

return size;

};

exports.memsize = function() {

var size = 0,

key;

for (key in cache) {

size++;

}

return size;

};

exports.debug = function(bool) {

debug = bool;

};

exports.hits = function() {

return hitCount;

};

exports.misses = function() {

return missCount;

};

exports.keys = function() {

return Object.keys(cache);

};

var cache = Object.create(null);

加载此模块时,初始化 cache 对象,

从上到下 首先来看一下 put api

exports.put = function(key, value, time, timeoutCallback) {

if (debug) {

console.log('caching: %s = %j (@%s)', key, value, time);

}

var oldRecord = cache[key];

if (oldRecord) {

clearTimeout(oldRecord.timeout);

} else {

size++;

}

var expire = time + Date.now();

var record = {

value: value,

expire: expire

};

if (!isNaN(expire)) {

var timeout = setTimeout(function() {

exports.del(key);

if (typeof timeoutCallback === 'function') {

timeoutCallback(key);

}

}, time);

record.timeout = timeout;

}

cache[key] = record;

};

接受4个参数 ,key,value,time(过期时间) , timeoutCallback (到期时回调函数)

var oldRecord=cache[key];

if(oldRecord)

如果此缓存已经存在:

clearTimeout(oldRecord.timeout);

这个方法大家应该明白, 清楚 setTimeout 定时任务的.(这里的作用就是,如果此缓存已经存在,接着你有调用了put 函数,说明你要更新此缓存,那么就把之前的定时器清除掉)

esle

size ++;

这里的 size 是维护当前缓存个数的一个变量,如果从来没有缓存过此对象,说明是新加的缓存,size 也相应加一.

var expire=time+Date.now();

过期时间=我们传入的时间间隔+当前时间.

var record 此对象包含了缓存 value 和 过期时间.

接着

if(!isNaN(expire)){

//如果设置了缓存时间,创建一个定时器(当定时器过期失效时,清楚此缓存)

var timeout=setTimeout(function(){

//调用 del() 函数清楚缓存

},time);

//record 对象上添加了一个属性 timeout

record.timeout=timeout;

}

//cache 对象添加了一个属性 key ,和 value ( record )

cache[key]=record;

看完这个大概就明白了.这个模块其实维护了一个对象 cache={};

添加一个 key='index\_page' value=[a,b,c,d,e,f,g]; 的缓存时.

相当于

cache={

'index\_page':

[a,b,c,d,e,f,g]

}

接着我添加一个带过期时间的缓存: 3000毫秒后过期 key='list' value=[1,2,3,4,5];

相当于

cache={

'index\_page':[a,b,c,d,e,f,g],

'list':{

timeout:当前时间+3000,

value:[1,2,3,4,5]

}

}

删除一个缓存,其实就是在 cache 对象上删除一个属性为 key 的字段.

如 : 删除 'index\_page' 缓存

delete cache.index\_page

如果我要更新 list 缓存:

首先要清楚 list 定义的定时器

clearTimeout(cache.list.timeout); //此timeout 就是定时器ID,根据ID可以直接清除掉定时器

接着判断新更新的缓存有没有设置过期时间

如果有,继续设置一个新的定时器.

如果没有,直接用最新的value 覆盖之前的 value 即可达到更新缓存效果.

非常轻量级,使用起来也非常简单!