Code traps in Nodejs
在Nodejs上踩过的坑

Who am I?我是谁？

Alibaba Data EDP

阿里巴巴数据平台EDP，花名@苏千

Chinese nodejs community: cnodejs.org

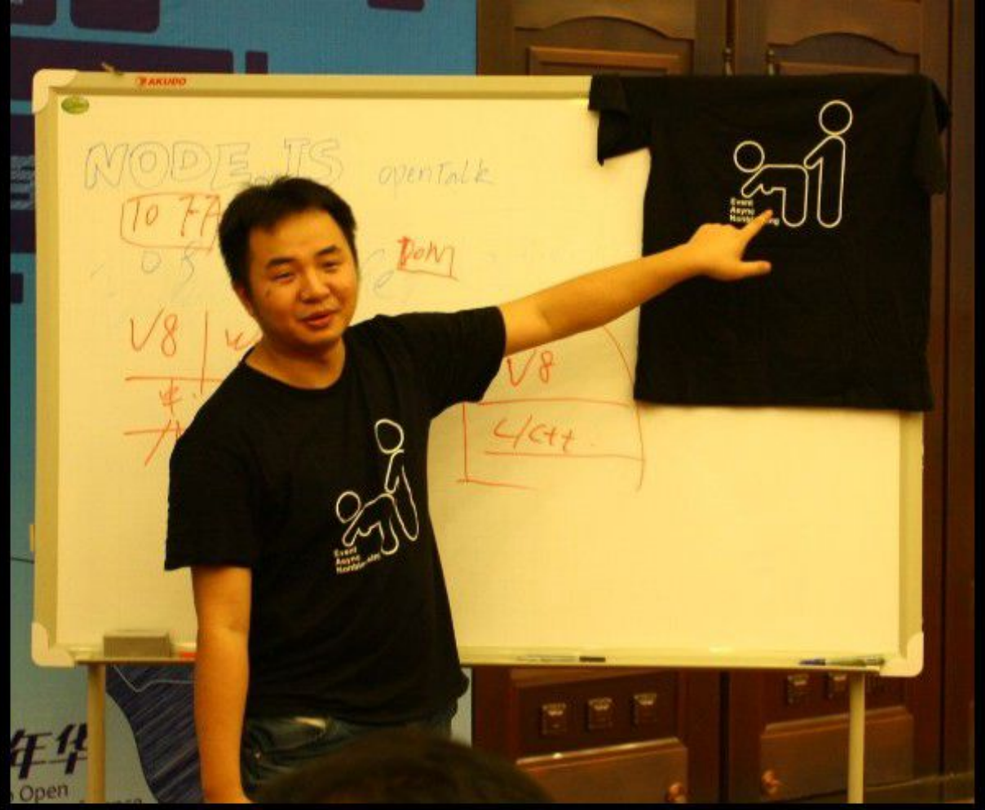
Github: @fengmk2

Blog: http://fengmk2.github.com

Twitter: @fengmk2

Weibo: @Python发烧友 , @FaWave

> 生命是一场幻觉



@nodejs workshop on TDC2011

绕开 => 绕不开 => 踩上去 => 享受

使用 nodejs 近一年来，遇到过许多千奇百怪的坑

一开始，遇到坑，第一反应是绕开
逐渐发现，有些坑，你是绕不过去的
然后，尝试去fixed它们
到现在，很喜欢未被人发现的坑，然后踩上去

Three traps | 三个坑

1. callback was called twice | 回调函数被调用两次
2. 40ms RT delayed | 响应延时了40毫秒
3. black hole in mongodb | mongodb驱动的黑洞

callback was called twice
回调函数被调用两次

> I won't write out this obvious bug.
> 我不会写出这种显而易见的bug。

callback was called twice
回调函数被调用两次

Really?
真的不会吗？

Can you find out the trap?
你能找出隐藏很深的坑吗？

Code come from  @TJ: connect-redis.js

```
RedisStore.prototype.get = function(sid, fn){
  sid = this.prefix + sid;
  debug('GET "%s"', sid);
  this.client.get(sid, function(err, data){
    if (err) return fn(err);
    try {
      if (!data) return fn();
      data = data.toString();
      debug('GOT %s', data);
      fn(null, JSON.parse(data));
    } catch (err) {
      fn(err);
    }
  });
};
```

try {} catch (e) {}

```
try {
  // ...
  fn(null, JSON.parse(data));
} catch (err) {
  fn(err);
}
```

What would happen when `fn(null, data)` throw an Error?

当 `fn(null, data)` 调用抛出异常的时候，会发生什么事情？

`fn` called twice

try {}

catch (err) {

}

```
fn(err);
```

was invoked

pull request for this trap

connect-redis#37

```
RedisStore.prototype.get = function(sid, fn){
  sid = this.prefix + sid;
  debug('GET "%s"', sid);
  this.client.get(sid, function(err, data){
    if (err) return fn(err);
    try {
      if (!data) return fn();
      data = data.toString();
      debug('GOT %s', data);
      data = JSON.parse(data);
    } catch (err) {
      return fn(err);
    }
    fn(null, data);
  });
};
```

Do not try catch the callback

不要捕获回调函数的异常

40ms RT delayed
响应延时了40毫秒

As we know, http.Agent don't not support really `keepalive`. So I wrote the agentkeepalive to support this feature.

我们知道，http.Agent 并不支持真正意义上的`keepalive`。所以我写了agentkeepalive模块支持这个特性。

When I using the agentkeepalive module on production environment, found out the *http response time(RT)* increase `40ms` unexpectedly.

当我在生产环境使用agentkeepalive模块的时候，发现*http 请求响应时间(RT)* 竟然增加了`40ms`。

agentkeepalive

build status passing

The nodejs's missing `keep alive` `http.Agent`.

jscoverage: 93%

Install

```
$ npm install agentkeepalive
```

Usage

```
var http = require('http');
var Agent = require('agentkeepalive');

var keepaliveAgent = new Agent({
  maxSockets: 10,
  maxKeepAliveRequests: 0, // max requests per keepalive socket, default is 0, no limit.
  maxKeepAliveTime: 30000 // keepalive for 30 seconds
});
```

What causes RT increase?
什么原因导致RT增长了？

Google "40ms delayed"

The answer is: Nagle algorithm and TCP delayed ack

答案是: Nagle算法和TCP延迟确认

`write-write-read` sequences will wait for a delayed ack timeout when Nagle's algorithm enabled.

当Nagle算法生效的时候，`write-write-read`这种方式将会等待一个延迟确认超时后，才会吧数据发送出去。

Why 40ms?
为什么是40毫秒呢？

> 2.13. Reducing the TCP delayed ack timeout: Some applications that send small network packets can experience latencies due to the TCP delayed acknowledgement timeout. This value defaults to 40ms.

We can even reducing the timeout value to 1ms by this:

我们甚至可以将超时时间设置为1毫秒:

```
# echo 1 > /proc/sys/net/ipv4/tcp_delack_min
```

Reappear the RT delay
重现RT延迟

Server: *nagle_delayed_ack_server.js*

```
require('http').createServer(function (req, res) {
  var start = Date.now();
  req.on('end', function () {
    res.end('hello world');
    console.log('[%s ms] %s %s',
      Date.now() - start, req.method, req.url);
  });
}).listen(1984);
```

Client: *nagle_delayed_ack_client.js*

```
// agentkeepalive@0.1.0: should reappear the delay problem
var Agent = require('agentkeepalive');
var agent = new Agent();
function request(callback) {
  var options = {port: 1984, path: '/fengmk2',
    method: 'POST', agent: agent};
  var start = Date.now();
  var req = require('http').request(options, function (res) {
    res.on('end', function () {
      console.log('[%s ms] %s',
        Date.now() - start, res.statusCode);
      callback();
    });
  });
  req.write('foo');
  process.nextTick(function () { req.end('bar'); });
}
function next() {
  setTimeout(function () { request(next); }, 1000);
}
next();
```

Run Server and Client

```
$ node nagle_delayed_ack_server.js
[1 ms] POST /fengmk2
[40 ms] POST /fengmk2 // increase 40ms
[38 ms] POST /fengmk2
[38 ms] POST /fengmk2

$ node nagle_delayed_ack_client.js
[7 ms] 200
[41 ms] 200 // increase 40ms
[40 ms] 200
[40 ms] 200
```

RT increase 40ms from the second request

从第二次请求开始，RT增加了40ms

pull request for agentkeepalive

commit@b04778071a9e2a5a47516daebe16c8f175b92460

Set socket.setNoDelay(true)

Disables the Nagle algorithm.
禁用Nagle算法即可解决问题

```
var s = http.Agent.prototype.createSocket.call(this, name, host,
    port, localAddress, req);
s.setNoDelay(true);
```

Read more: 模拟 Nagle 算法的Delayed Ack

Black hole in mongodb
mongodb驱动的黑洞

Everyone like mongodb.

大家都喜欢mongodb.

We connecting mongodb with node-mongodb-native and mongoskin.

我们使用node-mongodb-native 和 mongoskin 来连接mongodb.

Upgrade to mongodb@1.1.0
更新到mongodb@1.1.0

Query, Insert, Update are worked, but CPU load very high in ReplSet mode!

虽然查询，插入，更新等操作都正常工作，但是，CPU负载在ReplSet模式下意外地变得很高！

Google not work at this time.

即使Google一下也没找到头绪。

Explore the cause of the problem
探索问题原因

既然能正常工作，但是CPU很高，那么就代表着不是所有服务器都出问题了。

于是我人工连接每一台服务器测试，果然有发现： `Arbiter` 有异常。

```
$ telnet arbiter.mongodb.fengmk2.com

Trying arbiter.mongodb.fengmk2.com...
Connected to arbiter.mongodb.fengmk2.com.
Escape character is '^]'.
Connection closed by foreign host.
```

`Connected`, then `Closed` immediately.

Why connected -> closed?
为什么会出现连接后马上断开的情况呢？

I tell the situation to DBA, he gave me the answer : ACL

我将发现的情况咨询DBA，得到的回答是： ACL

ACL: Access control logic

ACL would blocked all unauthorized network access.

ACL会拦截掉所有未授权的网络访问。

因为DBA认为，我们只需要访问Primary和Secondanry，所以未开通对Arbiter的权限。

Is really causes by ACL?
真的是ACL导致的吗？

Mock the network ACL:

为了验证这个问题，我写了一个黑洞模拟代码：

```
var mongodb = require('mongodb');
var count = 0;
var blackhole = require('net').createServer(function (c) {
  console.log('new connection: ' + count++);
  c.end();
});
blackhole.listen(24008, function () {
  var replSet = new mongodb.ReplSetServers([
    new mongodb.Server('127.0.0.1', 24008, {auto_reconnect: true})
  ]);
  var client = new mongodb.Db('test', replSet);
  client.open(function (err, p_client) {
    console.log(err);
  });
});
```

$ node mongodb_blackhole.js

```
$ node mongodb_blackhole.js
new connection: 0
new connection: 1
new connection: 2
new connection: 3
...
new connection: 3326
new connection: 3327
new connection: 3328
new connection: 3329
new connection: 3330
new connection: 3331
```

CPU load high

PID	COMMAND	%CPU	TIME	#TH	#WQ	#POR	#MREG	RPRVT	R
1259	node	79.4	00:03.42	2/1	0	22	118	31M+	2
307	Terminal	47.7	00:31.12	7	3	156	303	22M+	3
0	kernel_task	33.3	02:26.87	82/2	0	2	905	82M	0
1255	top	8.4	00:04.17	1/1	0	26	31	1740K+	2

The problem is reappear.

问题总算被重现了。

Search the source code
从代码中找答案

I found out the problem, it eat the CPU by `/lib/mongodb/connection/connection_pool.js`.

经过一轮的代码阅读，我找到将CPU吃光的代码了：`/lib/mongodb/connection/connection_pool.js`。

Show me the code | 上代码吧

```
connection.on("connect", function(err, connection) {
  // Add connection to list of open connections
  _self.openConnections.push(connection);
  // If the number of open connections is equal to the poolSize
  if(_self.openConnections.length === _self.poolSize && _self._poolState !== 'disconnected') {
    // Set connected
    _self._poolState = 'connected';
    // Emit pool ready
    _self.emit("poolReady");
  } else if(_self.openConnections.length < _self.poolSize) {
    // need to open another connection, make sure it's in the next
    // tick so we don't get a cascade of errors
    process.nextTick(function() {
      _connect(_self);
    });
  }
});
```

when `openConnections.length < poolSize`, pool will create a new connection in `nextTick`.

But in meantime, if the openning connection `emit('close')`, this `openConnections.length` will be cleanup to `0`.

```
connection.on("close", function() {
  // If we are already disconnected ignore the event
  if(_self._poolState !== 'disconnected' && _self.listeners("close").length > 0) {
    _self.emit("close");
  }
  // Set disconnected
  _self._poolState = 'disconnected';
  // Stop
  _self.stop();
});
```

This case will be `Infinite loop`, open, close, open, close...

这就导致了死循环，不断地 open, close, open, close...

How to fixed this?

如何修复？

Waiting for your pull request on

node-mongodb-native

Conclusion 总结

通常，解决问题是一瞬间的事情

但是，真正理解问题所在，重现问题，给出最准确的解决方案，是一个非常漫长的过程。

码路上本有千奇百怪的坑，踩的人多了，

也便成了高速公路。

(@GoddyZhao, please help me to translate this.)

```
hujs.emit('Thanks')
&&
console.log('end');
```