

Nodejs 脱离了浏览器的Javascript

Who am I?

Twitter: @fengmk2

Weibo: @Python发烧友 , @FaWave

来自淘宝EDP, 花名@苏千

内容

1. Hello world 性能对比

为什么Nodejs会比其他语言的Hello world性能要好?

2. String = Buffer => Stream

从String到Buffer, 从Buffer到Stream, 文件数据流, 网络数据流

Javascript可以轻松地通过Socket实现各种网络协议, 还有功能完善的HTTP模块。

3. 脱离了浏览器的 Javascript

Nodejs能带来更多什么魔法呢?

4. 第三方模块

第三方模块的持续增长和完善, 让Nodejs的世界变得越来越有趣。

模块管理神器: npm

Hello world

性能对比

Nodejs的特点

- 单进程
- 异步非阻塞
- 事件驱动

Nodejs , Tornado , Go , Netty

- Nodejs: <http://nodejs.org>
- Tornado: <http://www.tornadoweb.org>
- Go: <http://golang.org>
- Netty: <http://www.jboss.org/netty>

Python: Tornado

```
from tornado.ioloop import IOLoop
from tornado.web import RequestHandler, \
    Application

class MainHandler(RequestHandler):
    def get(self):
        self.write("Hello, world")

application = Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    application.listen(8080)
    IOLoop.instance().start()
```

Go:

```
package main
import (
    "http"
    "io"
)
func handler(w http.ResponseWriter, r *http.Request) {
    io.WriteString(w, "Hello, world")
}
func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Java: Netty

```
private void handleHttpRequest(
    ChannelHandlerContext ctx,
    HttpRequest req) throws Exception {
    HttpResponse res =
        new DefaultHttpResponse(HTTP_1_1, OK);
    res.setHeader(CONTENT_TYPE,
        "text/html; charset=UTF-8");
    setContentLength(res, 11);
    res.setContent(
        ChannelBuffers.copiedBuffer(
            "hello world",
            CharsetUtil.US_ASCII));
    sendHttpResponse(ctx, req, res);
}
```

Nodejs:

```
var http = require('http');
http.createServer(function(req, res){
    res.end('Hello, World');
}).listen(8080);
```

Nodejs: 4CPU

```
var cluster = require('cluster')
, http = require('http');
var server =
  http.createServer(function(req, res) {
    res.end('Hello World');
});
cluster(server)
.set('workers', 4)
.use(cluster.stats())
.listen(8080);
```

测试环境

CPU:

```
$ cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
```

8 Intel(R) Xeon(R) CPU E5410 @ 2.33GHz

内存: 16GB

操作系统:

```
$ cat /etc/issue | grep Linux
```

Red Hat Enterprise Linux Server release 5.4 (Tikanga)

测试脚本

```
$ ab -c 30 -n 1000000 http://127.0.0.1:8080/
```

测试结果对比

Name	30 threads rps	100 rps	1000 rps	5000 rps
Nodejs	7,287	7,038	6,786	6,912
Tornado	2,967	2,966	2,967	2,982
Go	5,214	5,157	5,080	5,164
Netty	13,526	13,351	11,332	7,921
Nodejs 4P	14,826	14,332	12,161	8,287

** Nodejs性能很不错！ **

String
=> Buffer =>
Stream

Javascript数据处理的局限性

原生Javascript只有Unicode String

难以处理流数据

在处理TCP流和文件系统时经常需要操作字节流。

原生Javascript处理起来会很吃力。

V8 1GB 堆栈内存限制

大文件怎么办呢？处理的数据需要占用超过1GB
内存怎么处理呢？

Buffer

Buffer存储字节流数据，类似于一个整数数组。

解决：流数据处理问题

在 V8堆 (the V8 heap) 外的原始存储空间分配。

解决：V8 1GB 堆栈内存限制

一旦创建了Buffer实例，则无法改变其大小。
分配给进程的一块内存空间。

Buffer

```
// buffer.js
var a = new Buffer(10);
console.log(a, a.toString());
var b = new Buffer('QCon2011杭州');
console.log(b, b.toString());
```

```
$ node buffer.js
<Buffer 05 08 4e 00 2f 0f 26 05 04 4e>
'\u0005\bN\u0000/\u000f&\u0005\u0004N'
<Buffer 51 43 6f 6e 32 30 31 31 e6 9d ad e5 b7 9e>
'QCon2011杭州'
```

fs, net, http(s)

操作文件系统：`var fs = require('fs')`

文件的I/O是由标准POSIX函数封装而成。
所有的方法都提供了异步和同步两种方式。

网络操作：`var net = require('net')`

提供了一种异步网络包装器，
它包含创建服务器和客户端（称为streams）所需的方法(只提供异步方式)。

HTTP 服务以及客户端：

`var http(s) = require('http[s]')`

Node中的HTTP接口在设计时就考虑到了要支持HTTP协议的很多特性，并且使用简单。
特别是可以处理那些内容庞大，有可能是块编码的消息。
该接口被设计为从不缓冲整个请求或相应，这样用户就可以以流的方式处理数据。

fs, net, http(s)

fs的同步方法

```
var fs = require('fs');
// $ touch /tmp/helloSync
fs.renameSync('/tmp/helloSync',
'/tmp/worldSync');
var stats =
  fs.statSync('/tmp/worldSync');
console.log('statsSync: '
+ JSON.stringify(stats));
```

fs, net, http(s)

fs的异步方法

```
var fs = require('fs');
// $ touch /tmp/hello
fs.rename('/tmp/hello', '/tmp/world', function (err) {
  if (err) throw err;

  fs.stat('/tmp/world',
    function (err, stats) {
      if (err) throw err;
      console.log('stats: '
        + JSON.stringify(stats));
    });
});
```

fs, net, http(s)

最简单的telnet聊天室(1)

```
// chat.js
var net = require('net');
var clients = [];
net.createServer(function(client) {
  client.write('Enter your name:\n');
  client.once('data', function(data) {
    var username = data.toString().trim();
    clients.push(client);
    broadcast(username + ' : Join!\n');
    client.on('data', function(data) {
      var text = username + ' : ' + data;
      broadcast(text);
    });
  });
}).listen(11021);
```

fs, net, http(s)

最简单的telnet聊天室(2)

```
// 单进程的优势。。
function broadcast(text) {
    console.log(text.trim());
    var i = 0, l = clients.length;
    for(; i < l; i++) {
        var c = clients[i];
        c.write(text);
    }
};
```

fs, net, http(s)

最简单的telnet聊天室(3)

Server: \$ node chat.js

```
mk2 : Join!  
mk2 : Hello qcon2011 hangzhou!
```

Client: \$ telnet 192.168.1.xxx 11021

```
Enter your name:  
mk2  
mk2 : Join!  
Hello qcon2011 hangzhou!  
mk2 : Hello qcon2011 hangzhou!
```

fs, net, http(s)

http server

```
var http = require('http');
http.createServer(function(req, res) {
  if(req.url === '/') {
    res.end('Hello world');
  } else {
    res.end('HTTP ' + req.method
      + ' : ' + req.url);
  }
}).listen(8080);
```

fs, net, http(s)

http client

```
var http = require('http');
var options = {
  host: 'www.google.com',
  port: 80,
  path: '/'
};

http.get(options, function(res) {
  console.log("Got response: "
    + res.statusCode);
  res.on('data', function(data) {
    console.log(data.toString());
  });
}).on('error', function(e) {
  console.log("Got error: " + e.message);
});
```

Stream: 流的方式

Stream (流) 是一个由不同对象实现的抽象接口。

例如请求HTTP服务器的request是一个流，类似于stdout（标准输出）。
读取文件可以是一个文件流。

流可以是可读的，可写的，或者既可读又可写。

所有流都是EventEmitter的实例。

形象地如流水一般，数据在各个流之间可以按水流方向流动(Pipe)。

上传文件：fs.ReadStream

创建只读文件流

```
var readstream =  
    fs.createReadStream(uploadfile);
```

通过监听文件的data事件，获取数据，就像它自己会吐数据出来一样 而不用自己去调用read方法，一点一点地去取数据

上传文件：fs.ReadStream

```
readstream.on('data', function(chunk) {  
    console.log('write', chunk.length);  
    // 向服务器发送数据  
    req.write(chunk);  
});
```

通过end事件可以判断文件数据是否全部读取完了

```
readstream.on('end', function() {  
    req.end();  
});
```

Pipe：将水管连接起来

嫌既监听data又要监听end事件很麻烦？那就试试pipe吧，简直像安装水管那么简单。

直接使用pipe，想象两端水管，我们只需将他们按照水流方向连接起来即可（吐数据 ==> 收数据）
当数据读取完，会自动触发req.end()

```
readstream.pipe(req);
```

你没眼花，就是一行代码这么简单，所有数据就会自动发出去了。

通过readStream读取大文件并发送到网络中去：upload_file.js

Pipe: 将水管连接起来

:) 呵呵，原来

程序员

也是

水电工。

下载文件：fs.WriteStream

创建只写文件流

```
var writestream =  
  fs.createWriteStream(savefile);
```

继续做水电工，安装水管，还是以水流的方向安装
(吐数据 ==> 收数据) 这次网络流变成吐数据，
文件流变成收数据

```
res.pipe(writestream);
```

下载文件：fs.WriteStream

文件句柄已关闭， 回调结果

```
writestream.on('close', function() {  
  callback(null, res);  
});
```

通过WriteStream接收网络中得到的数据：download_file.js

程序员水电工：Pipe



脱离了浏览器的Javascript

Nodejs能带来更多什么魔法呢？

不受限制地访问文件和网络资源

忘记跨域访问的烦恼吧

前后端代码复用

服务端程序

桌面程序

...无限想象...

服务端程序

常规Web应用

实时性强的网络应用

网络中间层服务

.....

特别适合各类 网络**IO密集型** 应用

常规Web应用

淘job: 基于微博的企业化招聘网站



快乐工作在淘宝

用户

用微博账号登录

职位搜索

搜索

职位标签

共享数据平台(66)

Java(30)

共享数据平台_量子恒道(30)

实习生(20)

产品&运营(17)

C/C++(17)

共享数据平台_数据魔方(16)

Web开发(16)

Js/Html/Css/Flash(16)

共享数据平台_共享中心(11)

职位列表

数据开发工程师

挑战海量数据，发现数据价值。

由 @淘宝泽远 发布

收藏(1) 投递简历(0) 转发(3) 评论(3)

数据架构师

挑战海量数据，投身平台建设。

由 @淘宝泽远 发布

收藏(2) 投递简历(0) 转发(2) 评论(0)

Hadoop/Hive数据平台架构师

挑战海量数据，投身平台建设。

由 @淘宝泽远 发布

收藏(0) 投递简历(0) 转发(9) 评论(4)

开发平台产品经理

挑战海量数据，投身平台建设。

由 @淘宝泽远 发布

收藏(0) 投递简历(1) 转发(4) 评论(1)

实时性强的网络应用

Trello: 团队协作工具，
它的3个核心原则：实时更新，设备独立性，和最
小摩擦。

和美仪一起的日子

想做的

日本之旅

uncle

广州塔摩天轮

国庆大学城

五月天演唱会

Add card

去过的

万象城

沃尔玛

西湖

上海金融中心

同济大学

上海磁悬浮

美仪家姐屋企

0/2

烧烤

踩单车

国庆广州美食节

Add card

吃过的

绿茶

外婆家

两岸咖啡

一锅两吃

Add card

睇过的电影

iMax 变形金刚3

五月天3DNA 追梦

Add card

纪念日

七月初一 美仪生日

九月二七 我生日

Add card

Members

[Add Members...](#)

Board

[Add List](#)[Filter Cards...](#)[View Archive...](#)[Preferences...](#)

Activity [View all...](#)

added 五月天演唱会 to 想做的
10/7/2011

moved 国庆广州美食节 from 想做的 to 去过的
10/7/2011

moved 踩单车 from 想做的 to 去过的
10/7/2011

moved 烧烤 from 吃过

实时性强的网络应用

踩单车

in list 去过的



Vote

Members

No members assigned

Assign...

Attachments

No attachments

Upload

Actions

Add checklist

Add due date...

Labels...

Move...

Archive

实时性强的网络应用

PaintChat: 可以协作绘图的聊天室。

工具



线

重画 保存

铅笔

直径

10

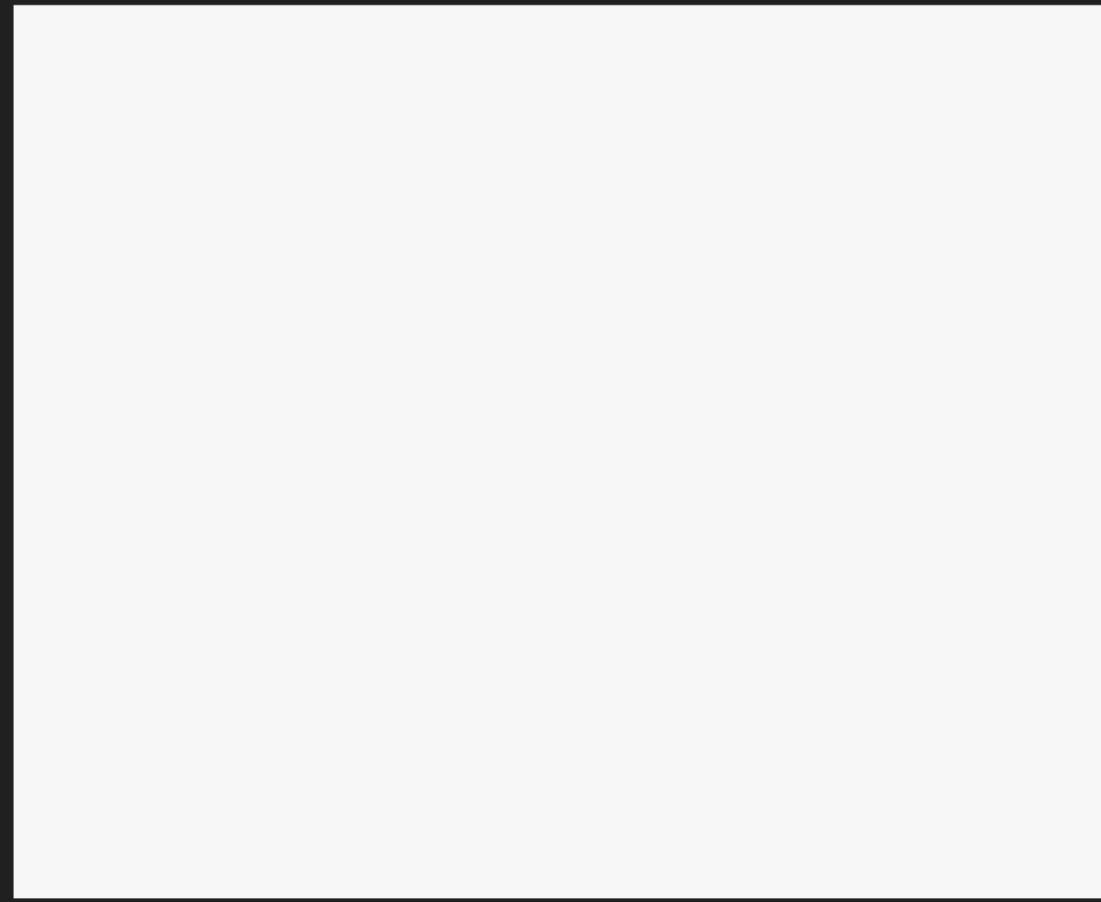
透明度

80

调色板



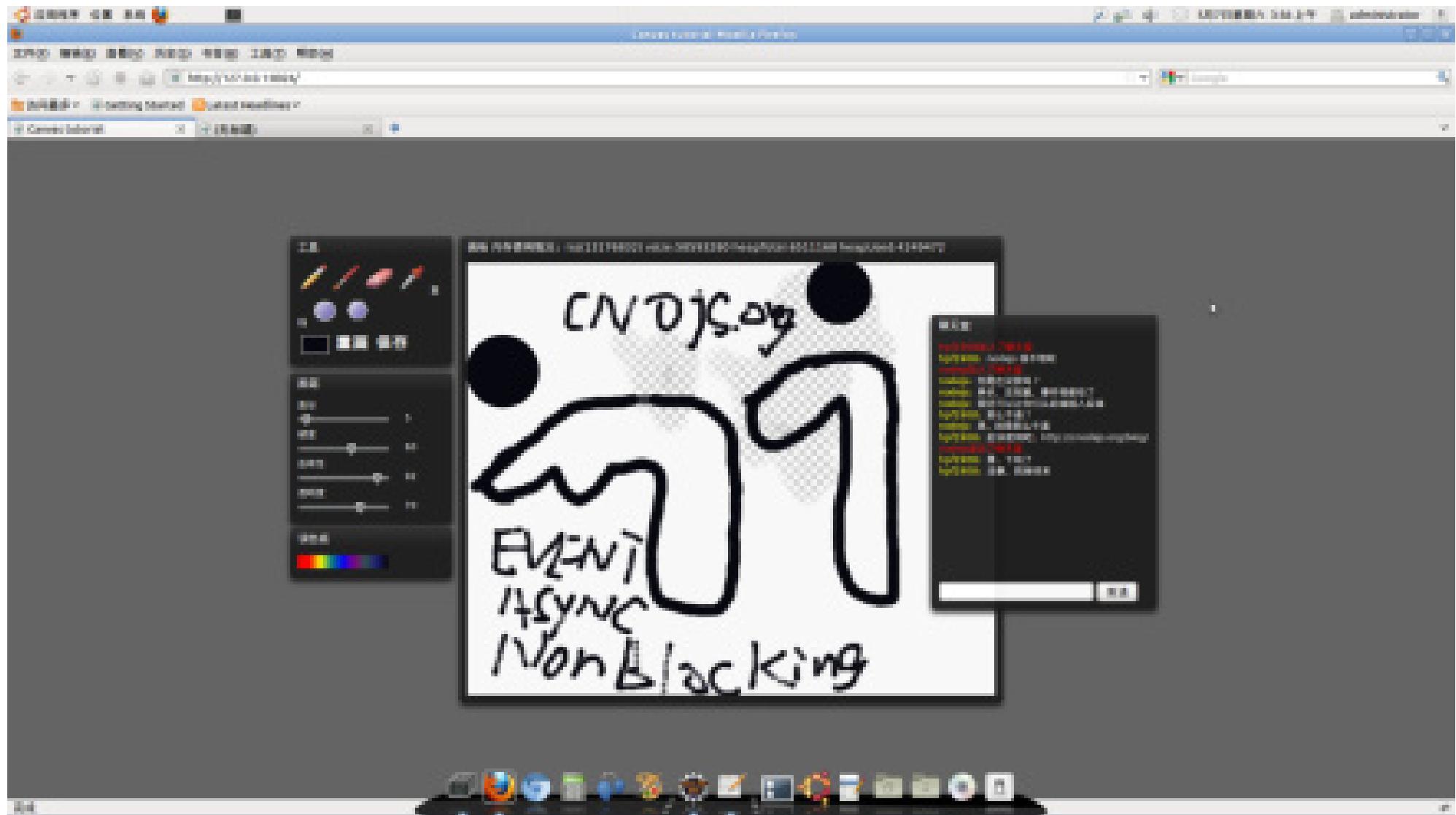
画板



聊天室

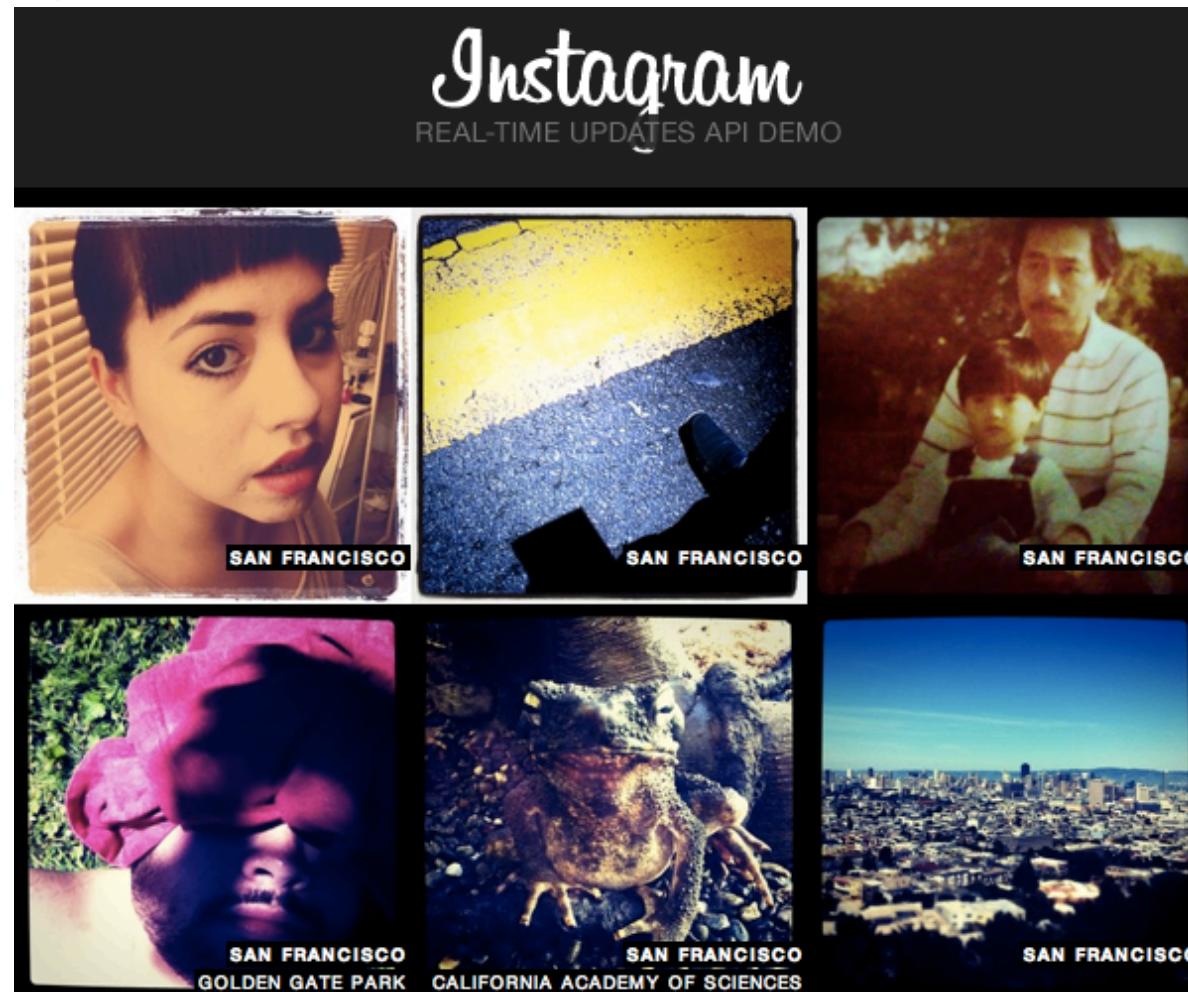
发送

实时性强的网络应用



实时性强的网络应用

Instagram Real-time API Demo: Instagram 官方给出的一个实时应用，服务器实时推送最新上传的照片到浏览器上显示。基于**Socket.io**模块实现的。



实时性强的网络应用

Cloud9: Web IDE，随时随地写代码。

File Edit View Windows | Save Open... debug run stop II Q L

Cloud9 IDE

File Manager dockpanel.js nodedebugproxy.js index.js

Project Files Preferences

Version 0.2.1

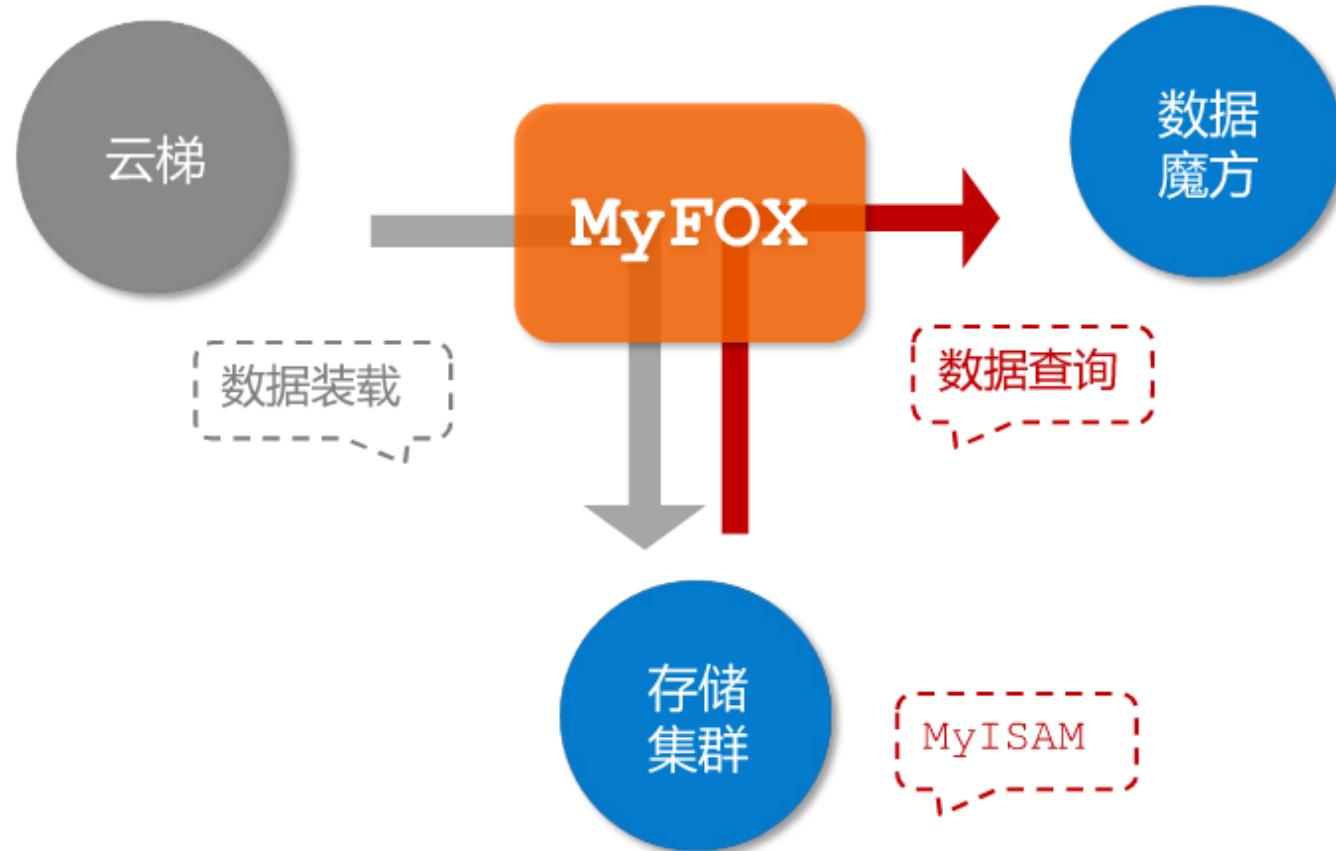
File Manager

- include
- js
- style
- favicon.ico
- require.js
- server
 - cloud9
 - dav
 - ext
 - auth
 - debugger
 - git
 - settings
 - shell
 - state
 - watcher
 - test
 - view
 - error.js
 - ide.js
 - index.js
 - middleware.js
 - netutil.js
 - optparse.js
 - plugin.js
 - template.js

```
1 /**
2 * @copyright 2010, Ajax.org Services B.V.
3 * @license GPLv3 <http://www.gnu.org/licenses/gpl.txt>
4 */
5 var net = require("net");
6 var sys = require("sys");
7 var NodeSocket = require("v8debug/NodeSocket");
8 var StandaloneV8DebuggerService = require("v8debug/StandaloneV8DebuggerService");
9
10 module.exports = DebugProxy = function(port) {
11     process.EventEmitter.call(this);
12     var _self = this;
13
14     this.connected = false;
15
16     var socket = new NodeSocket("localhost", port);
17     socket.onend = function() {
18         this.connected = false;
19         _self.emit("end");
20     };
21     this.service = new StandaloneV8DebuggerService(socket);
22
23     this.service.addEventListener('connect', function() {
24         _self.connected = true;
25         _self.emit("connection");
26     });
27     this.service.addEventListener('debugger_command_0', function(msg) {
28         _self.emit("message", msg.data);
29     });
30 };
31
32 sys.inherits(DebugProxy, process.EventEmitter);
33
34 (function() {
```

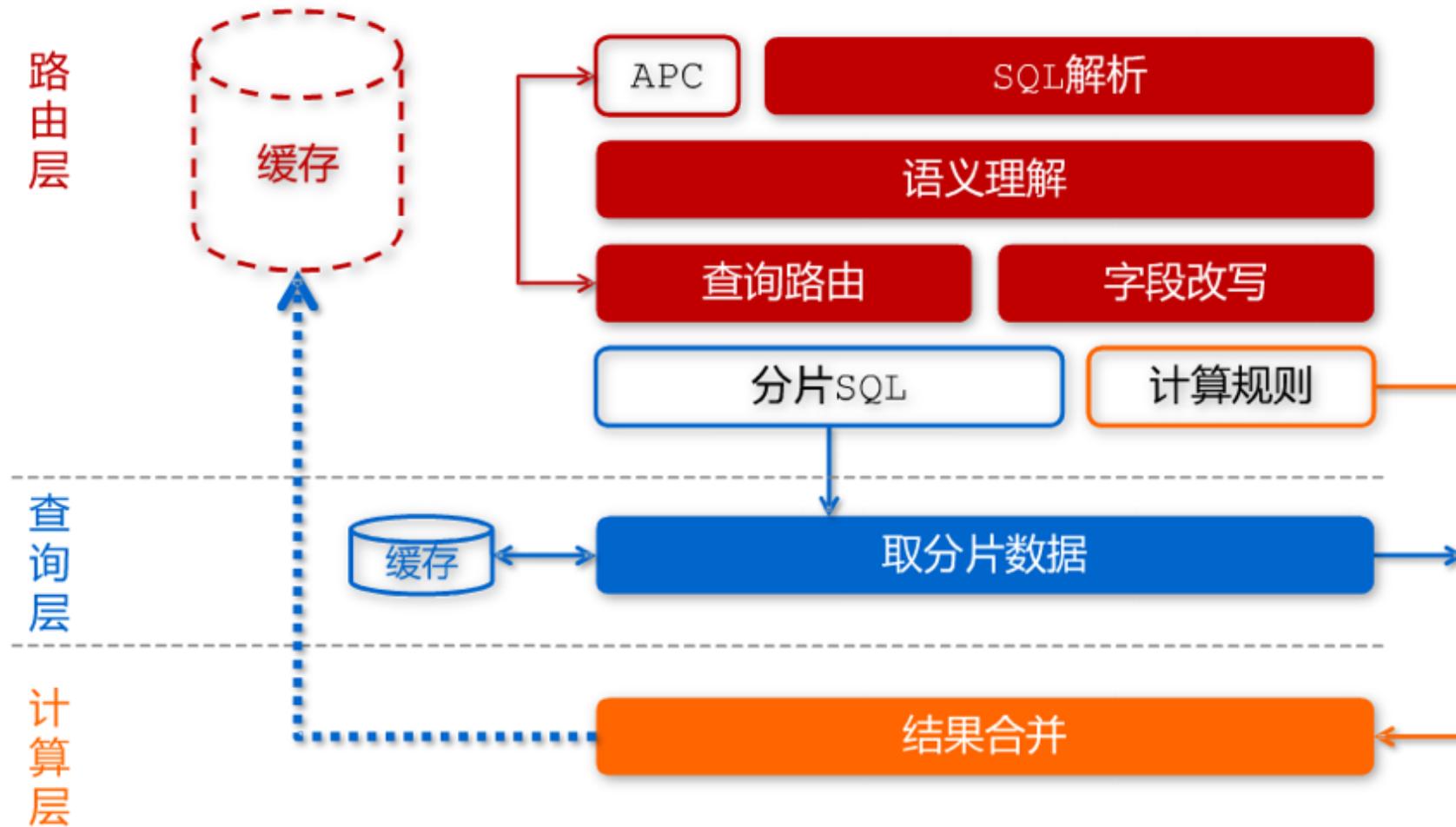
网络中间层服务: Nodefox

Nodefox: 一个数据处理中间件，负责从一个MySQL集群中提取数据，计算，并输出统计结果。



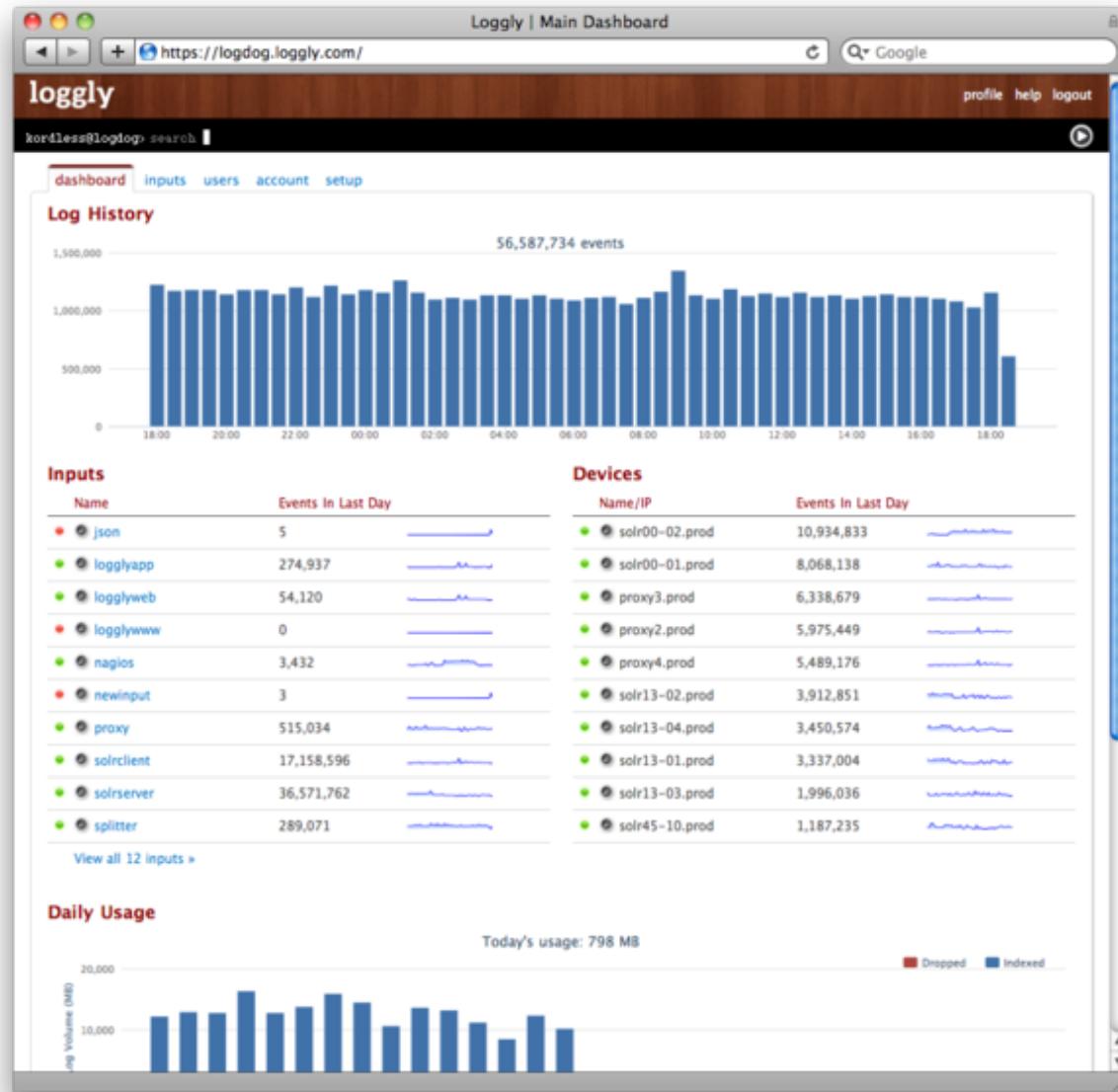
网络中间层服务: Nodefox

Nodefox的查询过程



网络中间层服务: Loggly

Loggly: 收集syslog和HTTP日志的Web服务。



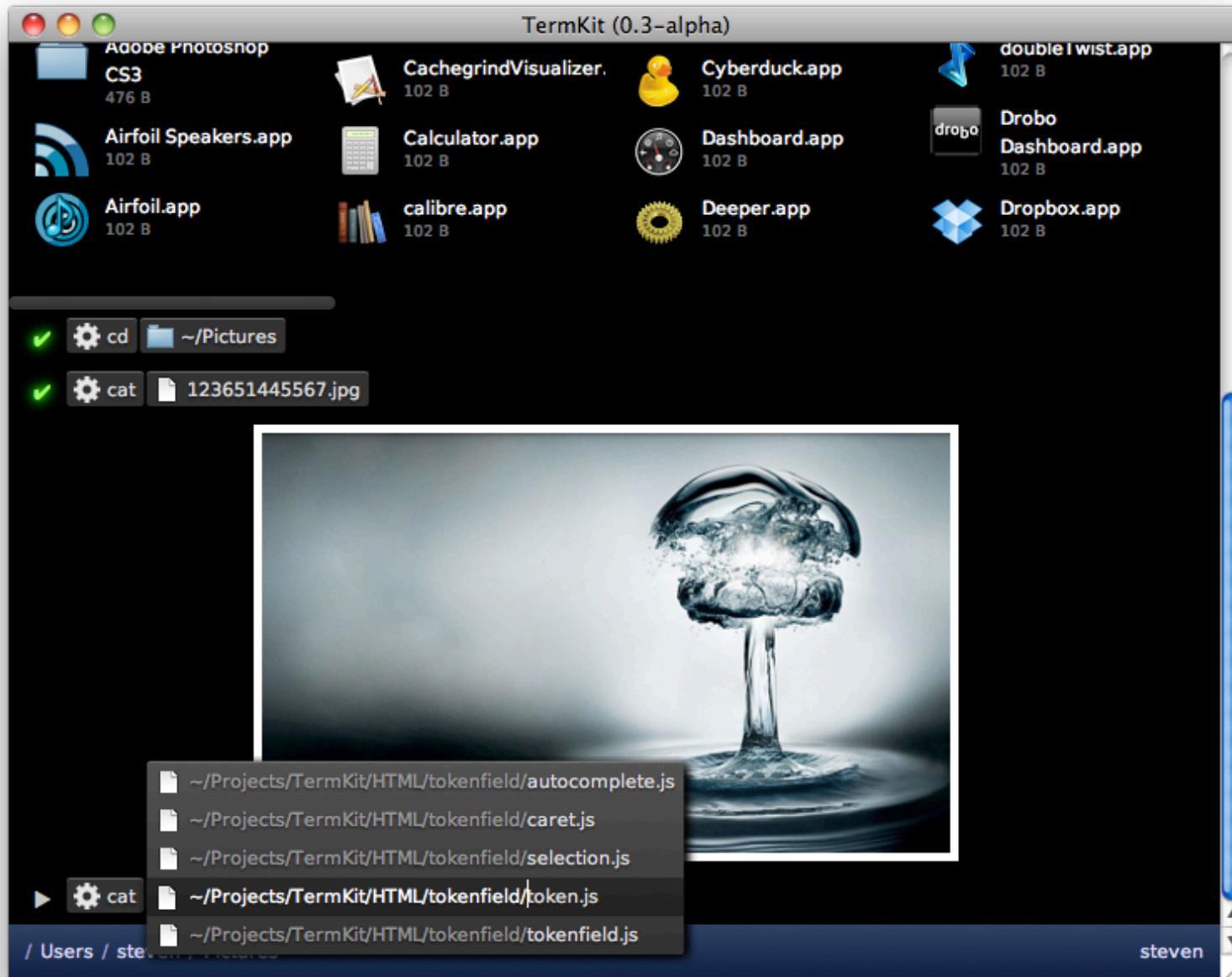
网络中间层服务

NAE Proxy: Node App Engine的Http proxy， 将应用的端口映射到本地socket。

app	▲ rss	◆ heap	◆ uptime	◆ last	◆ pid	◆ autorun	◆ running	◆ ports
q3boy1	17870848	7400688	336111.965632	10/15 23:45:13	9656	true	true	80
q3boy2	17838080	7362384	336111.87218299997	10/15 23:45:13	9657	true	true	80
qchat	22036480	10784624	336113.343203	10/15 23:45:14	9632	true	true	80
query	21204992	10365288	336112.56891599996	10/15 23:45:13	9636	true	true	80
quick	20623360	9637344	275758.558525	10/15 23:45:13	17984	true	true	80
qwchat	-1	-1	-1	10/12 02:23:19		false	false	
remindme	21979136	11379472	336116.840203	10/15 23:45:13	9461	true	true	8877
renfake	19800064	8896456	19756.637363999966	10/15 23:45:13	29576	true	true	80
renfake1	-1	-1	-1	10/15 17:36:59		false	false	
richie	17956864	7338512	336115.554843	10/15 23:45:13	9520	true	true	80

桌面程序: TermKit

TermKit: next gen terminal / command application, built out of WebKit and Node.js.



桌面程序: TermKit

TermKit (0.3-alpha)

✓ cat Desktop/temp/file.diff

```
1 --- a 2011-05-16 03:26:41.000000000 -0700
2 +++ b 2011-05-16 03:26:49.000000000 -0700
3 @@ -1,6 +1,9 @@
4 -sdfk jsdlkfj sldkfj
5 -skdfjsdlk jflksd
6 -sdfsdfsfdsdf
7 -sdfsd
8 +sdfsdfs fsd fsdfsd
9 +dfs
10 +dfsd
11 +f
12 +sdf
13 +sd
14 +fsd
15 -fsdf
16 +f
17 +sd
```

✓ cat Desktop/temp/test.xml

```
1 <?xml version="1.0" charset="utf-8" ?>
2 <root>
3   <bin></bin>
4   <bin attr="foo"></bin>
5 </root>
```

▶

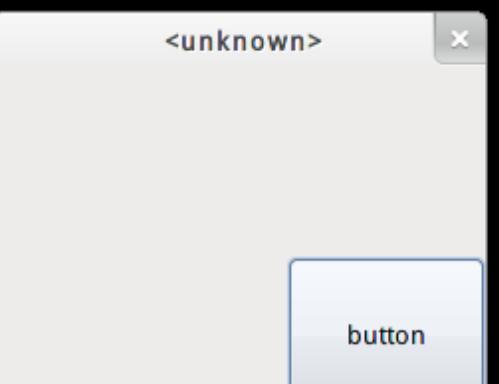
/ Users / steven steven

桌面程序: node-gui

@小型笨蛋 在尝试使用 nodejs驱动GTK+ demo

Terminal

```
1 var Builder = require('gui').Builder;
2
3 new Builder('./data/test.glade', function (builder) {
4     var window = builder.get ('window1');
5     window.show ();
6 });
~
~
```



第三方模块

Nodejs官方收集的第三方模块页面上，共列出了

|| 52

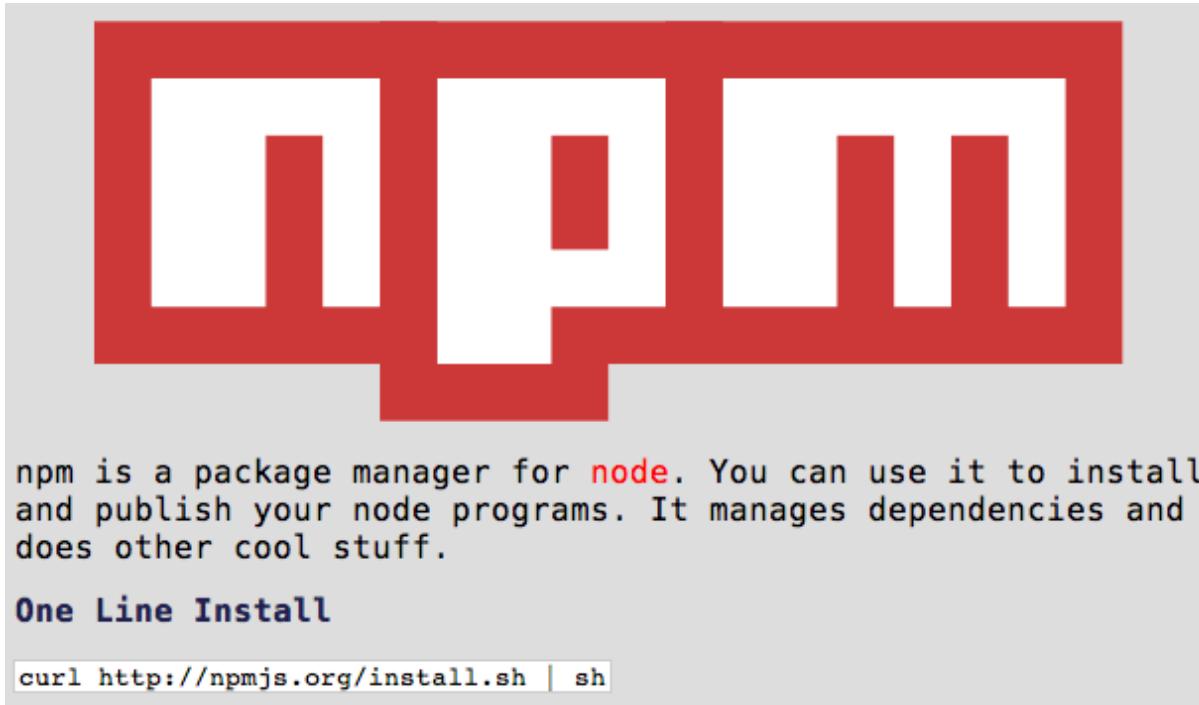
个模块。

第三方模块

涵盖了

Web, Database, Templating, CSS Engines, CMS, Build and Deployment, Openssl, Hashing, SMTP, TCP/IP, RPC, Web Sockets & Ajax, Message Queues, Testing, JSON, XML, Debugging, Compression, Graphics, Payment Gateways, API clients, Control flow / Async goodies, I18n and L10n modules 等等，几乎涉及了网络开发中需要的功能模块。

模块管理工具: npm



<http://npmjs.org/>

模块管理工具: npm

在npm上目前管理着

4427+

个nodejs模块， 只需要简单一行命令， 即可安装你需要得模块， 例如我想使用nodejs访问mysql数据库：

```
$ npm install mysql
```

常用模块(I)

- 中间件Middleware: Connect 是一个可扩展的 HTTP server框架， 提供了许多高效的插件。
- Web框架: Express, webjs
- 数据库驱动：
node-mysql, redis-node,
node-mongoskin

常用模块(2)

- Web Sockets & Ajax, RPC:
Socket.io, nowjs

- 单元测试：
nodeunit, expresso

- 流程控制/解决异步嵌套：
EventProxy, Jscex, async

Node's goal

Provide an **easy** way to build scalable
network programs.

提供一种**便捷**
的方式构建网络程序。

Q & A

知乎者也

```
next("Thanks ^_^")
```