



打造安全易运维的高性能Web平台

——淘宝网Nginx定制开发实战

朱照远（叔度）

王晓哲（清无）

2011-12-06

大纲

- 背景介绍
- 前端优化
- 安全增强
- 典型业务模块
- 运维增强
- 动态脚本与数据库层

1、背景介绍

淘宝网面临的的技术挑战

- 亚洲最大的电子商务网站，Alexa排名12
- 巨大的商品量
 - 商品总数超过10亿
 - 每天在线的商品数超过5亿
 - 每天更新的商品数超过6000万
- 巨大的访问量
 - 每秒钟几百个G的网络流量
 - 日PV超过几十亿的业务线很多

淘宝网使用Nginx的过程

- 2009年开始使用和探索
- 2010年开始开发大量模块
 - 基础的
 - 业务的
- 2011年开始
 - 修改Nginx的内核
 - [Tengine](#)项目并开源

The logo for Tengine, featuring the word "Tengine" in a stylized, blue, 3D-effect font with a white outline and a slight shadow.

淘宝网应用Nginx的收益

- 业务更加稳定
 - Nginx大连接数目支持非常好
 - Nginx本身的内存占用很少，更不会吃swap
- 业务性能更高
 - QPS比Apache要好
 - 节省机器数目
 - 基于Nginx的模块性能往往是之前业务的数倍

2、前端优化

组合JavaScript和CSS文件

- Yahoo!前端优化第一条原则
 - Minimize HTTP Requests
 - 减少三路握手和HTTP请求的发送次数
- 淘宝CDN combo
 - concat模块
 - 将多个JavaScript、CSS请求合并成一个

淘宝CDN Combo的使用

- 以两个问号 (??) 激活combo特性
- 多个文件之间用逗号 (,) 分开
- 用一个?来表示时间戳
 - 突破浏览器缓存
- 例子

```
http://a.tbcdn.cn/??s/kissy/1.1.6/kissy-min.js,p/global/1.0/global-min.js,p/et/et.js?t=2011092320110301.js
```

强制gzip压缩

- 来自Google的最佳实践
 - 2009、2010年Velocity大会
- 做强制gzip的原因
 - 代理软件、杀毒软件对Accept-Encoding头的修改或删除
 - 访问淘宝的请求 > 15% 没有带Accept-Encoding头

强制gzip的基本原理

- 如果请求中没有Accept-Encoding头或不支持gzip且没有GZ的cookie设置
- 判断浏览器（User-Agent）是否支持gzip
- 发送的内容中插入JavaScript脚本
- 脚本请求一个永远都gzip的URL
- 如果gzip的内容被执行了，说明支持gzip
- 设置GZ对应的cookie值，注明支持gzip

3、安全增强

单机安全方案

- 连接数限制
 - 使用limit_conn模块
- 访问频率限制
 - 加强版的limit_req模块
 - 白名单支持
 - 指定跳转页面支持
 - 同一个location下多limit_req支持

系统过载保护

- 判断依据
 - 系统的loadavg
 - 内存使用（swap的比率）
 - QPS
- sysguard模块

```
sysguard on;  
sysguard_load load=4 action=/high_load.html;  
sysguard_mem swapratio=10% action=/mem_high.html
```

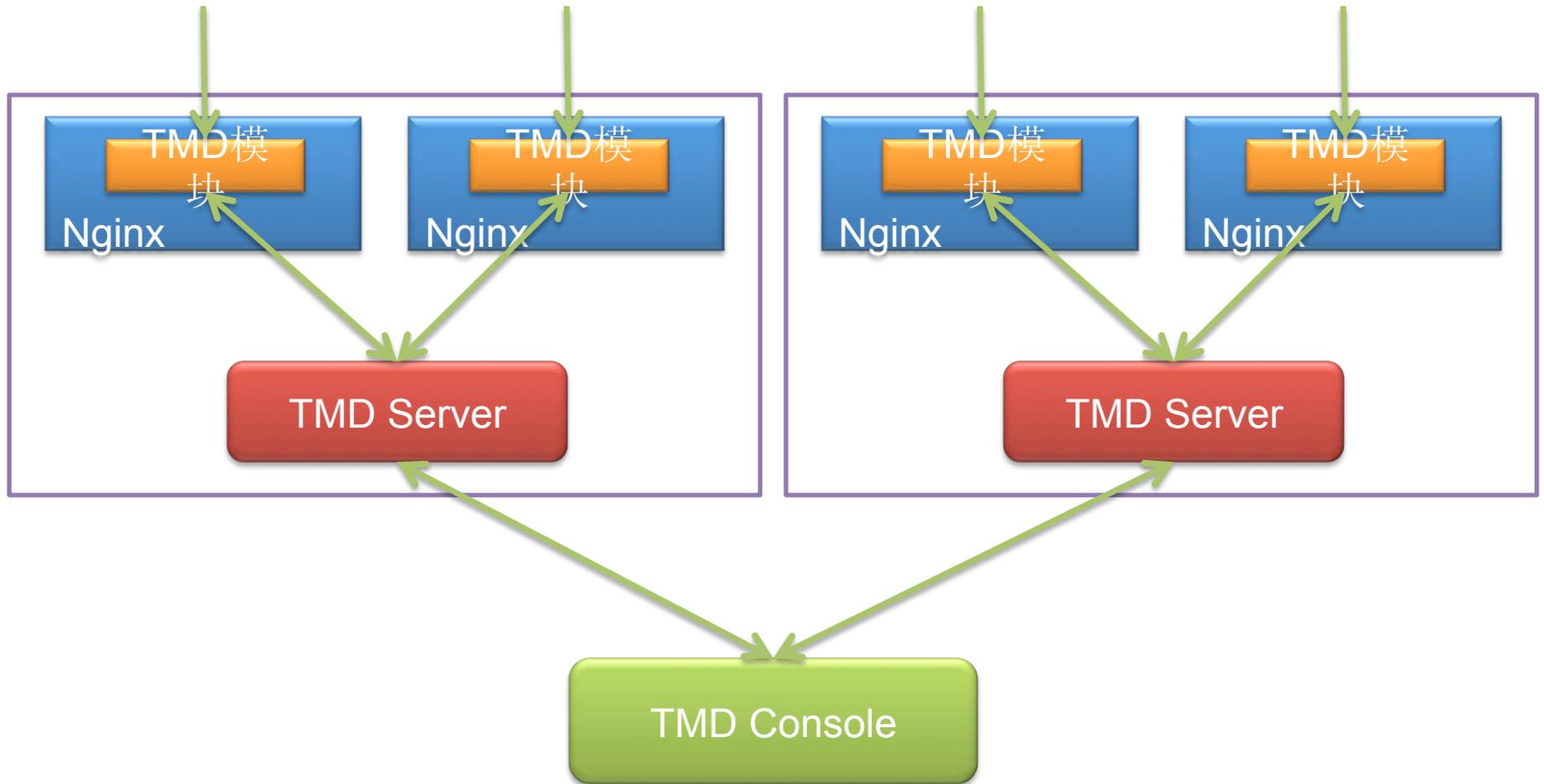
过载保护的等待页面

- 过载处理方式
 - 直接弹回（用户不友好，而且没有保护作用）
 - 返回等待页面（用户友好，有保护）
- 等待页面的处理
 - 将用户原来的请求内容返给用户（脚本）
 - 定时器倒计时
 - 时间到了自动发起新的请求

分布式防攻击系统

- 应对的问题
 - 小型的DDoS攻击
 - 恶意的爬虫
- 为什么单机版还不够
 - 单机版无法知道全局
- 淘宝TMD (Taobao Missile Defense) 系统
 - Nginx作为防攻击系统的终端
 - TMD Server做策略分析
 - TMD Console执行汇总和控制台

TMD防攻击系统架构

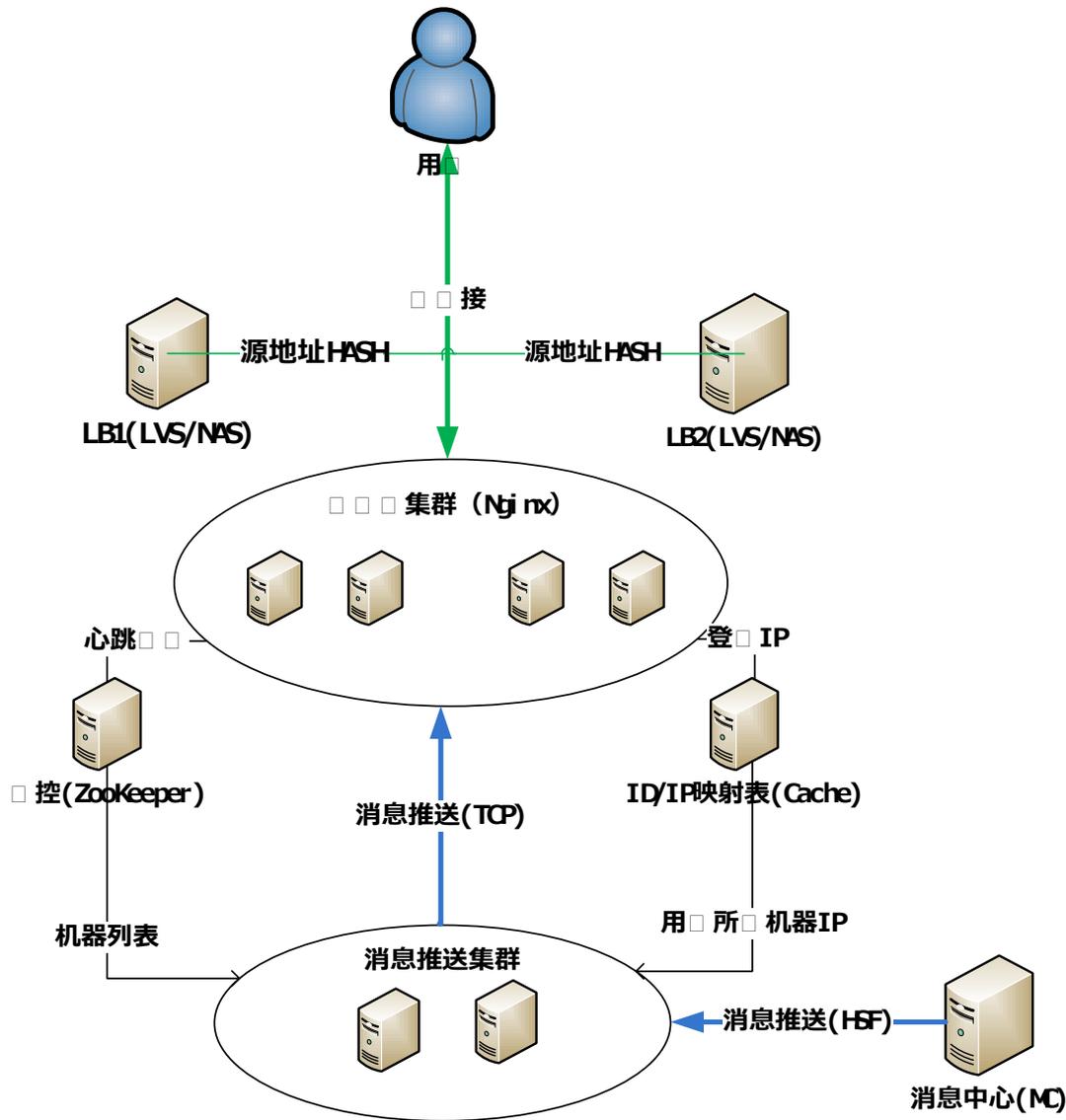


4、典型业务模块

Comet服务器

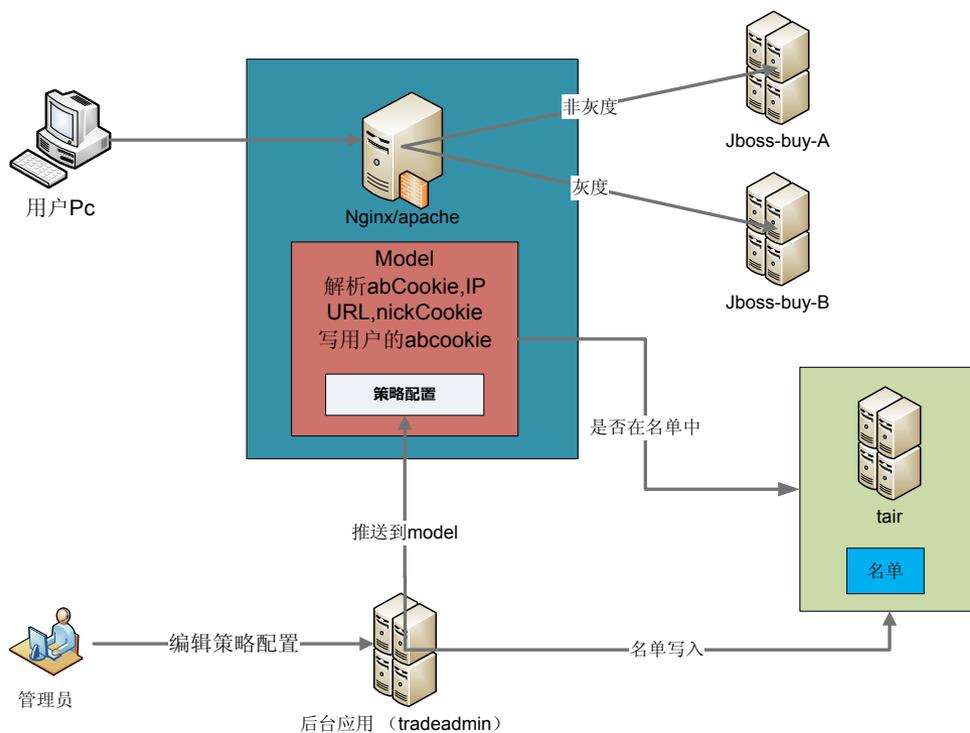
- Nginx本身的极限测试
 - 单机支持200万连接以上
- 淘宝消息推送系统
 - 部署容量60万连接/台
 - 实际跑到30万连接/台

Comet服务器架构



灰度发布

- 逐渐放量
- 方便的管理接口



淘宝开源分布式文件系统方案

- TFS模块

- 非upstream机制
- RESTful接口
- 简化了分布式存储方案的使用难度



淘宝开源分布式K/V存储方案

- [Tair](#)模块
 - 非upstream机制
 - RESTful接口
- 特点
 - 性能很高
 - 灵活
 - 返回JSON格式给客户端



5、运维增强

多种日志方式

- 本地和远程syslog支持

```
access_log      syslog:user:info:127.0.0.1:514 combined;
```

- 管道支持

```
access_log      pipe:/path/to/cronolog combined;
```

- 抽样支持

– 减少写日志的数量，避免磁盘写爆

```
access_log      /path/to/file combined ratio=0.01;
```

Server头的伪装

- 伪装

```
server_tag    Apache/2.2.21;
```

```
$ curl -I http://localhost
HTTP/1.1 200 OK
Server: Apache/2.2.21
Date: Sun, 04 Dec 2011 18:36:53 GMT
Content-Type: text/html
Content-Length: 205
Last-Modified: Tue, 01 Nov 2011 05:18:57 GMT
Connection: keep-alive
```

Server头的隐藏

- 隐藏

```
server_tag    off;
```

```
$ curl -I http://localhost
HTTP/1.1 200 OK
Date: Sun, 04 Dec 2011 18:41:16 GMT
Content-Type: text/html
Content-Length: 205
Last-Modified: Tue, 01 Nov 2011 05:18:57 GMT
Connection: keep-alive
```

主机信息调试

- Tengine的footer模块

```
footer $host_comment;
```

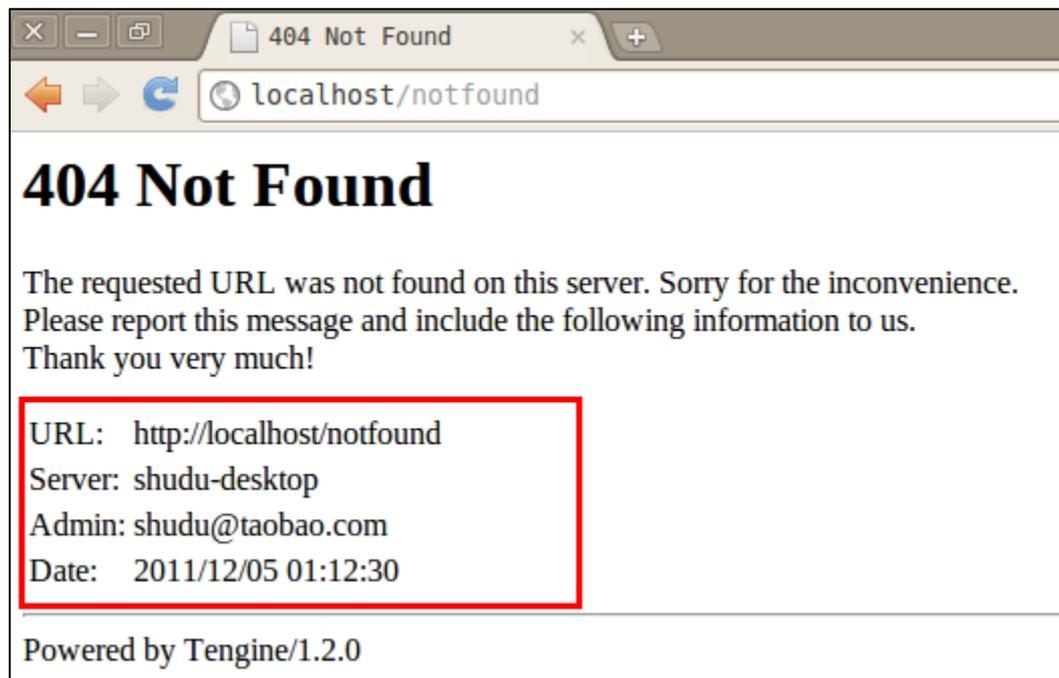
- 输出效果

```
$ curl http://localhost
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
</body>
</html>
<!-- shudu-desktop Sat, 03 Dec 2011 09:27:47 GMT -->
```

Tengine错误信息提示

- 便于定位用户反馈的4xx和5xx错误

```
server_info    on;  
server_admin  shudu@taobao.com;
```



worker进程和CPU亲缘性

- 好处
 - 利用多核
 - 防止CPU的cache失效
- 问题
 - 不同的硬件，CPU核数可能不同
 - 绑定多核的CPU亲缘性比较繁琐

Tengine对于进程设置的简化

- 使用对比

```
# standard nginx  
worker_process      8;  
worker_cpu_affinity 00000001 00000010 00000100 00001000  
                   00010000 00100000 01000000 10000000
```

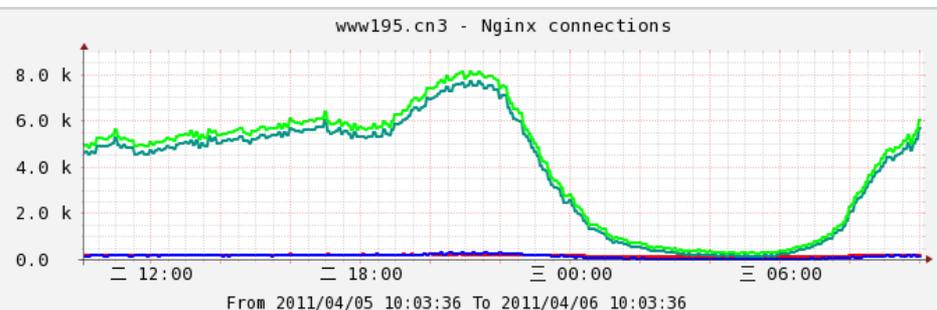


```
# tengine  
#worker_process      auto;  
#worker_cpu_affinity auto;
```

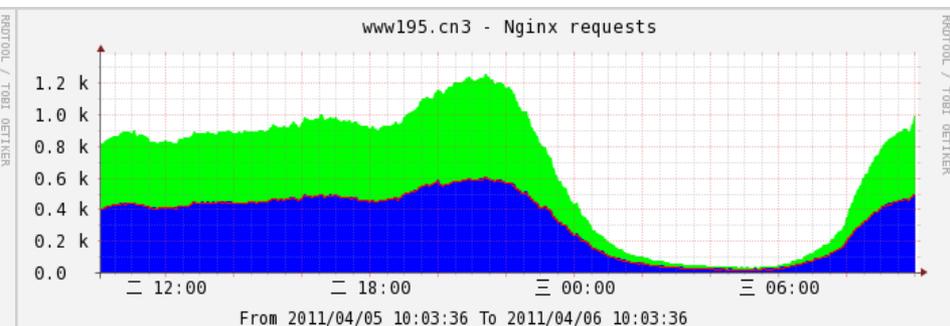
Nginx命令行参数的增加

- 列出已经编译的模块
 - nginx -m
- 列出支持的指令
 - nginx -l
- 输出配置文件的全部内容
 - nginx -d
 - 支持include的内容

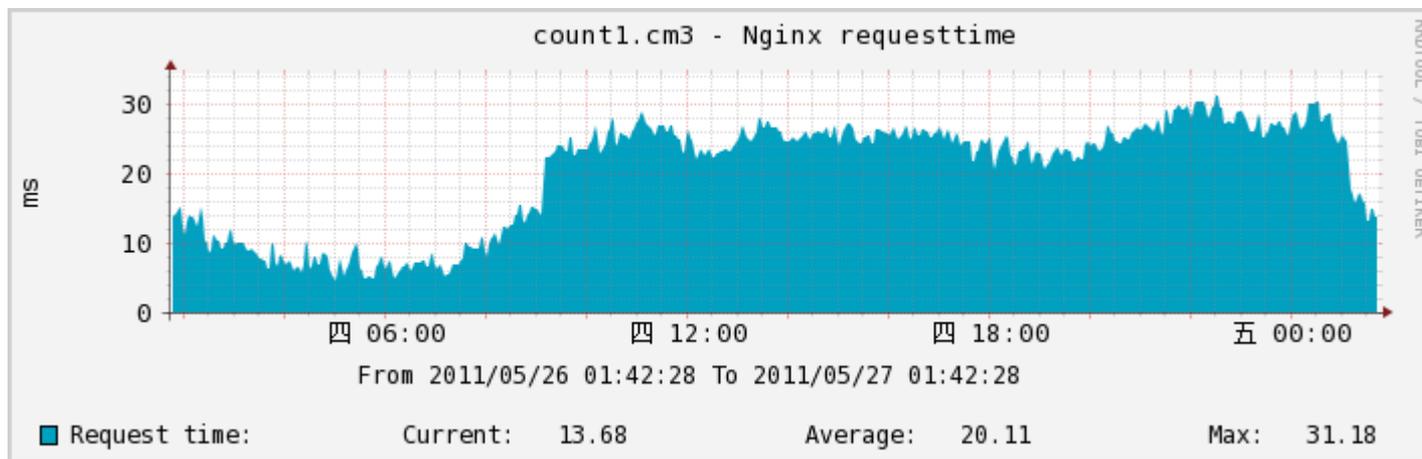
Nginx监控增强



Writing	Current: 184.99	Average: 167.74	Max: 222.86
Waiting	Current: 5.68 k	Average: 3.72 k	Max: 7.71 k
Reading	Current: 139.03	Average: 127.77	Max: 286.76
Total	Current: 6.01 k	Average: 4.02 k	Max: 8.12 k



Requests	Current: 987	Average: 648.84	Max: 1.25 k
Accepts	Current: 496	Average: 328.43	Max: 601.10
Handled	Current: 496	Average: 328.43	Max: 601.10



Request time:	Current: 13.68	Average: 20.11	Max: 31.18
---------------	----------------	----------------	------------

使用淘宝开源监控工具Tsar

- tsar --nginx

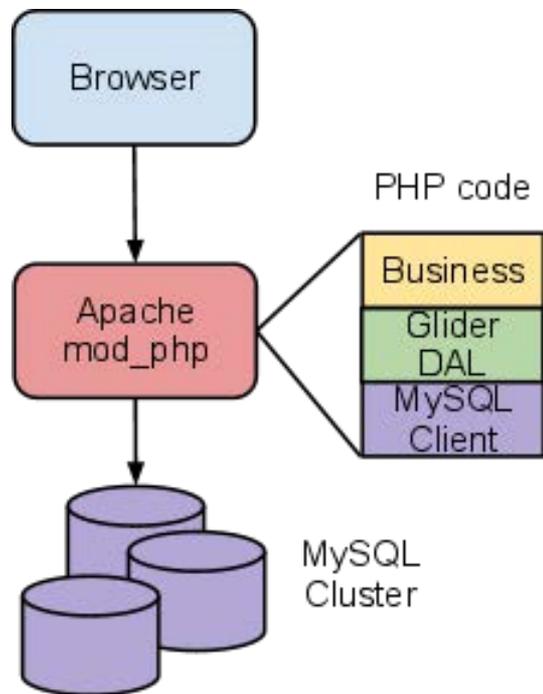
Time	accept	handle	reqs	active	read	write	wait	qps	rt
25/03-09:10	224.5K	224.5K	512.4K	14.5K	228.0	3.0	14.5K	1.7K	18.2
25/03-09:15	228.6K	228.6K	515.7K	14.6K	210.0	2.0	14.4K	1.7K	18.0
25/03-09:20	231.6K	231.6K	525.4K	15.1K	232.0	3.0	14.9K	1.8K	20.2
25/03-09:25	236.7K	236.7K	536.7K	15.2K	202.0	3.0	15.0K	1.8K	20.9
25/03-09:30	238.2K	238.2K	536.6K	15.3K	231.0	3.0	15.1K	1.8K	20.3
25/03-09:35	239.8K	239.8K	537.0K	15.3K	213.0	4.0	15.1K	1.8K	19.8
25/03-09:40	227.2K	227.2K	505.5K	14.0K	192.0	1.0	13.8K	1.7K	21.2
25/03-09:45	227.2K	227.2K	505.5K	1.0	0.0	1.0	0.0	1.7K	21.2
25/03-09:50	206.7K	206.7K	366.2K	10.2K	236.0	1.0	9.9K	1.2K	19.4
25/03-09:55	261.1K	261.1K	478.5K	10.6K	252.0	3.0	10.4K	1.6K	21.2
25/03-10:00	268.8K	268.8K	496.4K	11.1K	270.0	2.0	10.8K	1.7K	23.4
25/03-10:05	278.5K	278.5K	509.3K	11.2K	250.0	3.0	11.0K	1.7K	24.5
25/03-10:10	283.9K	283.9K	512.2K	11.5K	257.0	7.0	11.2K	1.7K	23.2
25/03-10:15	283.0K	283.0K	509.6K	11.3K	268.0	2.0	11.0K	1.7K	22.9
25/03-10:20	285.7K	285.7K	510.0K	11.4K	291.0	2.0	11.1K	1.7K	21.6
25/03-10:25	285.4K	285.4K	509.7K	11.3K	282.0	5.0	11.1K	1.7K	24.1
25/03-10:30	286.7K	286.7K	512.1K	11.4K	276.0	5.0	11.2K	1.7K	25.7
25/03-10:35	288.3K	288.3K	517.3K	11.4K	244.0	1.0	11.2K	1.7K	25.8
25/03-10:40	290.9K	290.9K	515.7K	11.7K	319.0	2.0	11.4K	1.7K	24.7

6、动态脚本与数据库层

背景

- 淘宝量子统计业务快速发展
 - 原架构无法满足业务需要
- 量子统计页面和数据特征
 - 页面主体框架基本不变
 - 查询复杂速度慢
 - 结果集数据量大
 - 重复查询少

思考

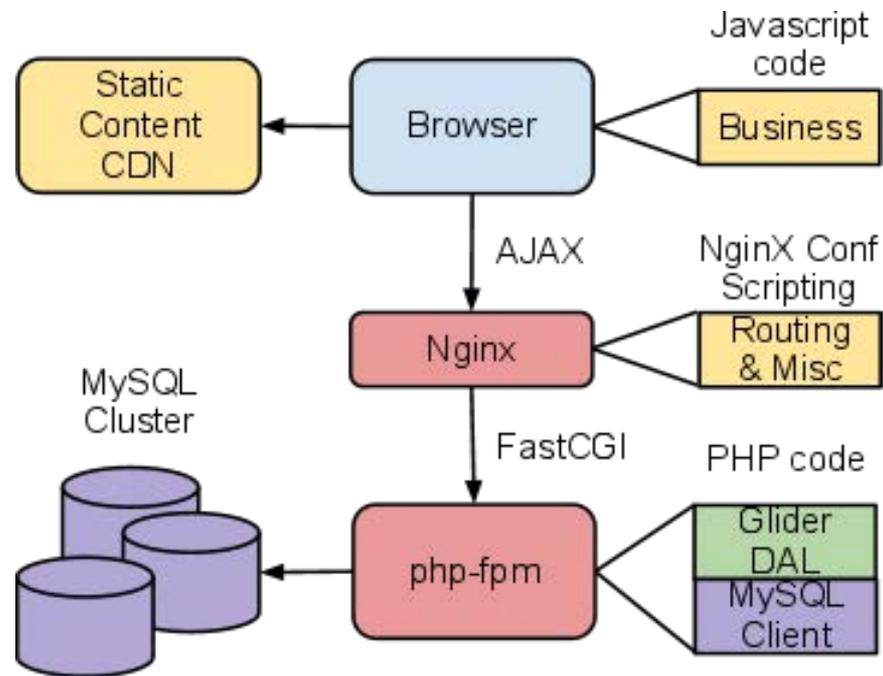
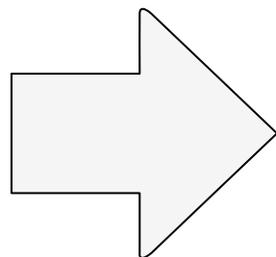
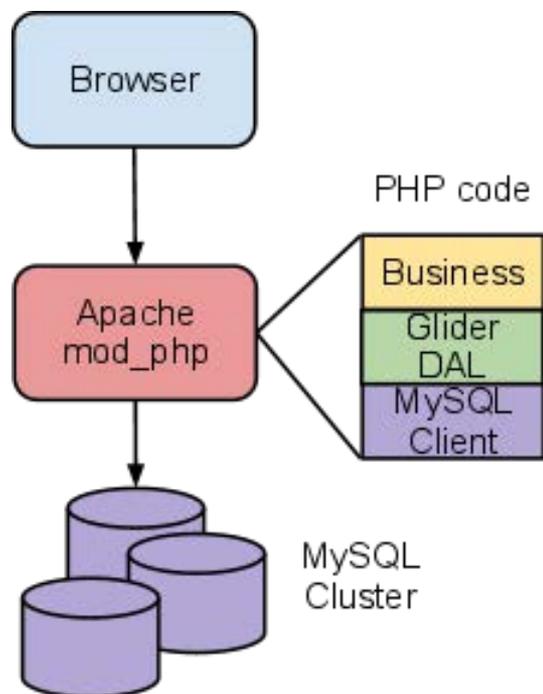


- 优点
 - 易于理解，开发上手快
- 缺点
 - 页面主体内容重复浪费带宽
 - 单机并发服务能力极为有限
 - 存在慢连接攻击风险
 - PHP代码处理大数据时速度低下

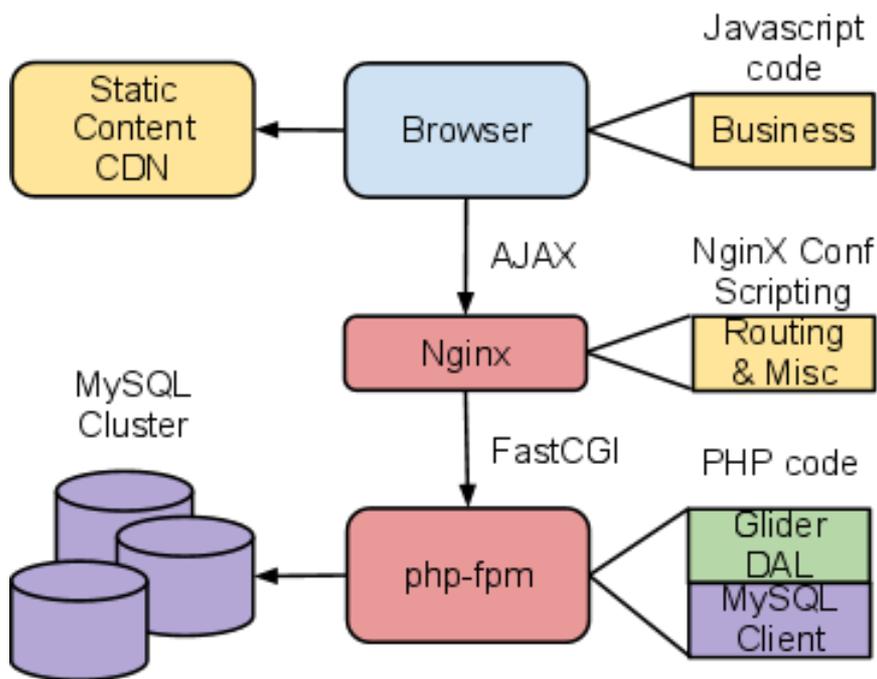
方案

- 更改服务模型
- 静态资源CDN化
- 服务侧页面组装过程移至浏览器侧

演进



思考



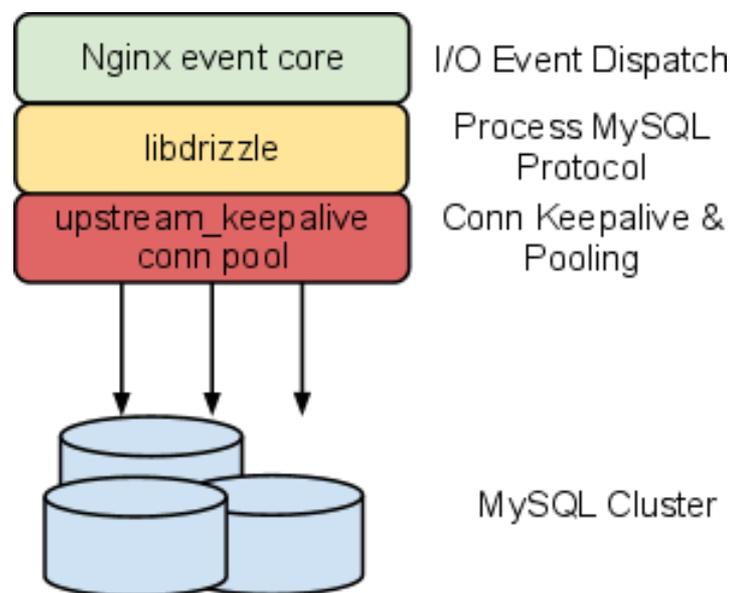
- 优点
 - 避免带宽浪费
 - 减少服务端计算量
 - 提升并发服务能力，可抵御慢连接攻击
- 缺点
 - 开发效率受限
 - PHP+FastCGI限制整体吞吐量提升

方案

- 要在Nginx中高效访问MySQL数据库
 - ngx_drizzle诞生!
- 要有适用Nginx I/O模型的高速脚本引擎
 - ngx_lua诞生!

ngx_drizzle

- 实现Nginx中同步非阻塞方式访问MySQL
- 具备长连接、进程级可控大小连接池和负载均衡功能
- 返回数据可通过ngx_rds_json/csv等模块转换为JSON/CSV格式



ngx_drizzle示例

```
http {  
    ...  
    upstream dbgroup {  
        drizzle_server host1:3306 dbname=test password=some_pass user=alice protocol=mysql;  
        drizzle_server host2:3306 dbname=test2 password=some_pass user=bob protocol=mysql;  
    }  
    ...  
    server {  
        location /mysql {  
            set $sql "select * from cats";  
            drizzle_query $sql;  
            drizzle_pass dbgroup;  
            rds_json on;  
        }  
    }  
}
```

Nginx C模块构建业务逻辑的问题

- 开发效率低
- 部署灵活性差

但是.....

- 人人都喜欢脚本语言!
- ngx_lua用Lua脚本构建业务逻辑!

Why Lua?

- 内存开销小
- 运行速度快
- VM可中断/重入

compare 2	-	---	25%	median	75%	---	-
<input type="checkbox"/> C GNU gcc	1.00	1.00	1.04	1.09	1.35	1.83	3.80
<input checked="" type="checkbox"/> C++ GNU g++	1.00	1.00	1.00	1.11	1.24	1.55	1.55
<input type="checkbox"/> ATS	1.00	1.00	1.00	1.26	1.57	2.42	7.19
<input type="checkbox"/> Java 6 steady state	1.00	1.00	1.15	1.57	1.96	2.06	2.06
<input checked="" type="checkbox"/> Lua LuaJIT	1.03	1.03	1.83	1.98	8.73	10.51	10.51
<input type="checkbox"/> Scala	1.14	1.14	1.19	2.00	2.39	4.19	6.67
<input type="checkbox"/> Fortran Intel	1.00	1.00	1.56	2.18	6.38	10.54	10.54
<input checked="" type="checkbox"/> Java 6 -server	1.10	1.10	1.36	2.18	3.63	6.80	6.80
<input type="checkbox"/> Ada 2005 GNAT	1.00	1.00	1.40	2.22	3.66	7.05	7.44
<input type="checkbox"/> Pascal Free Pascal	1.36	1.36	1.89	2.35	3.50	4.77	4.77
<input type="checkbox"/> Haskell GHC	1.14	1.14	1.92	2.38	3.97	7.04	8.08
<input type="checkbox"/> Clean	1.29	1.29	1.93	2.67	5.76	9.58	9.58
<input type="checkbox"/> C# Mono	1.69	1.69	1.97	2.84	5.16	9.95	18.38
<input type="checkbox"/> OCaml	1.49	1.49	2.52	3.04	3.87	4.62	4.62
<input type="checkbox"/> F# Mono	1.93	1.93	1.99	3.22	3.95	6.88	10.54
<input type="checkbox"/> Lisp SBCL	1.00	1.00	2.47	3.29	7.11	12.44	12.44
<input type="checkbox"/> Racket	1.45	1.45	2.84	4.38	8.01	15.77	19.20
<input type="checkbox"/> Go 6g 8g	2.65	2.65	3.70	4.66	12.40	25.43	125.45
<input checked="" type="checkbox"/> JavaScript VB	1.00	1.00	4.26	7.19	20.52	44.89	102.86
<input type="checkbox"/> Erlang HIPE	1.64	1.64	5.60	7.70	20.91	34.93	34.93
<input type="checkbox"/> Clojure	1.55	1.55	4.68	11.47	15.59	29.58	29.58
<input checked="" type="checkbox"/> JavaScript TraceMonkey	1.73	1.73	4.39	12.66	28.53	64.75	898.17
<input type="checkbox"/> Smalltalk VisualWorks	11.27	11.27	12.34	15.73	26.42	47.54	71.80
<input type="checkbox"/> Java 6 -Xint	7.37	7.37	14.91	23.85	32.36	58.53	72.81
<input checked="" type="checkbox"/> Lua	1.03	1.03	21.82	31.53	41.70	51.54	51.54
<input type="checkbox"/> Python PyPy	13.90	13.90	26.29	32.29	47.29	78.80	122.93
<input type="checkbox"/> Ruby JRuby	16.50	16.50	23.95	43.08	153.45	225.45	225.45
<input type="checkbox"/> Python CPython	2.28	2.28	5.74	47.96	93.11	106.47	106.47
<input type="checkbox"/> Python 3	2.23	2.23	7.10	50.09	128.75	157.73	157.73
<input type="checkbox"/> Python IronPython	20.41	20.41	33.78	58.41	86.05	164.46	190.75
<input type="checkbox"/> Mozart/Oz	7.57	7.57	32.47	63.37	85.64	165.40	197.18
<input type="checkbox"/> Ruby 1.9	7.64	7.64	15.33	63.66	106.91	244.27	265.59
<input type="checkbox"/> Perl	2.39	2.39	9.48	73.69	154.39	218.38	218.38
<input type="checkbox"/> PHP	6.90	6.90	48.32	104.15	149.90	233.84	233.84
<input type="checkbox"/> Ruby MRI	15.28	15.28	24.72	158.27	481.13	837.28	837.28

原理

- ngx_lua实现Proactor模型
 - 业务逻辑以自然逻辑书写
 - 自动获得高并发能力
 - 不会因I/O阻塞等待而浪费CPU资源

原理

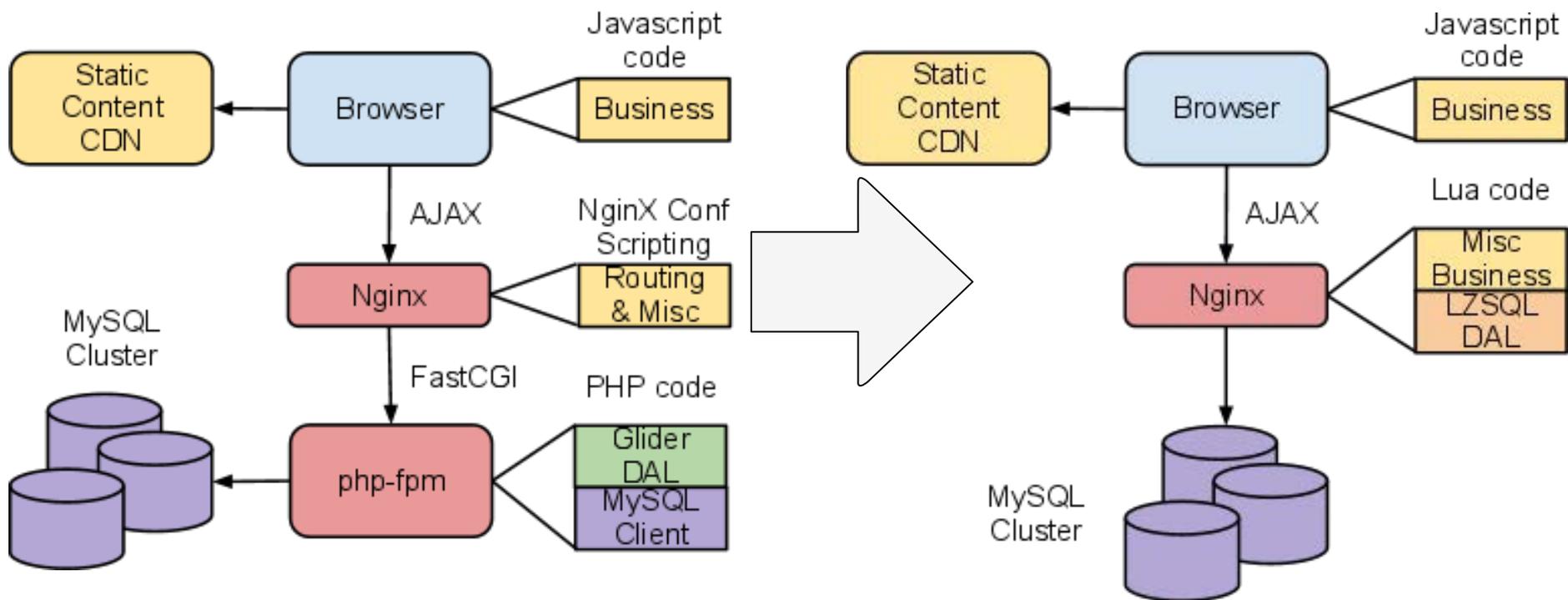
- 每Nginx工作进程使用一个Lua VM，工作进程内所有协程共享VM
- 将Nginx I/O原语封装后注入Lua VM，允许Lua代码直接访问
- 每个外部请求都由一个Lua协程处理，协程之间数据隔离
- Lua代码调用I/O操作接口时，若该操作无法立刻完成，则打断相关协程的运行并保护上下文数据
- I/O操作完成时还原相关协程上下文数据并继续运行

```
resolver ip.to.dns.server;
```

```
location /http_client {  
    internal;  
    proxy_pass $arg_url;  
}
```

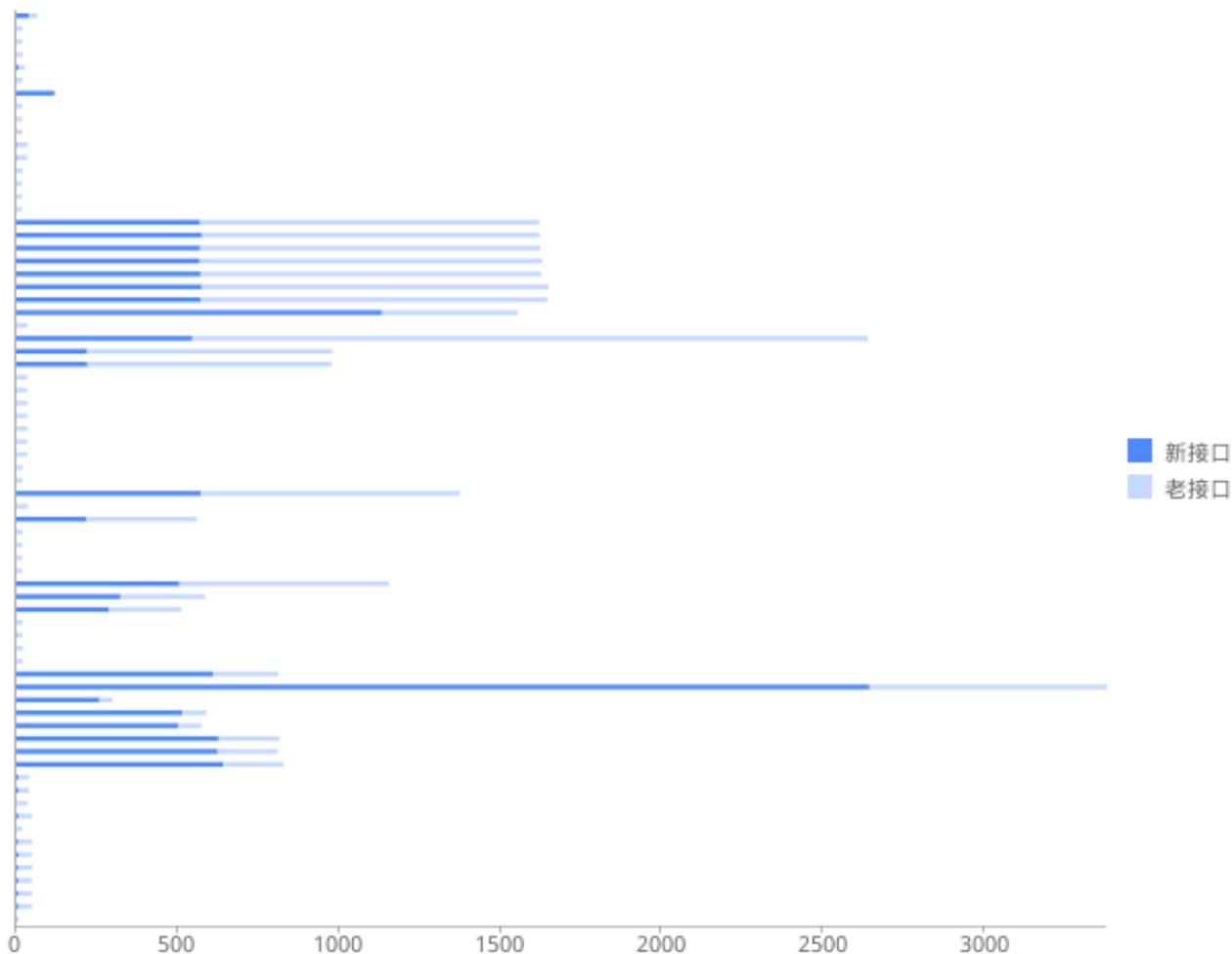
```
location /web_iconv {  
    content_by_lua '  
        local from, to, url = ngx.var.arg_f, ngx.var.arg_t, ngx.var.arg_u  
        local capture = ngx.location.capture  
        local iconv = require "iconv"  
        local cd = iconv.new(to or "utf8", from or "gbk")  
        local res = capture("/http_client?url=" .. url)  
        if res.status == 200 then  
            local ostr, err = cd:iconv(res.body)  
            ngx.print(ostr)  
        else  
            ngx.say("error occurred: rc=" .. res.status)  
        end  
    ;  
}
```

演进

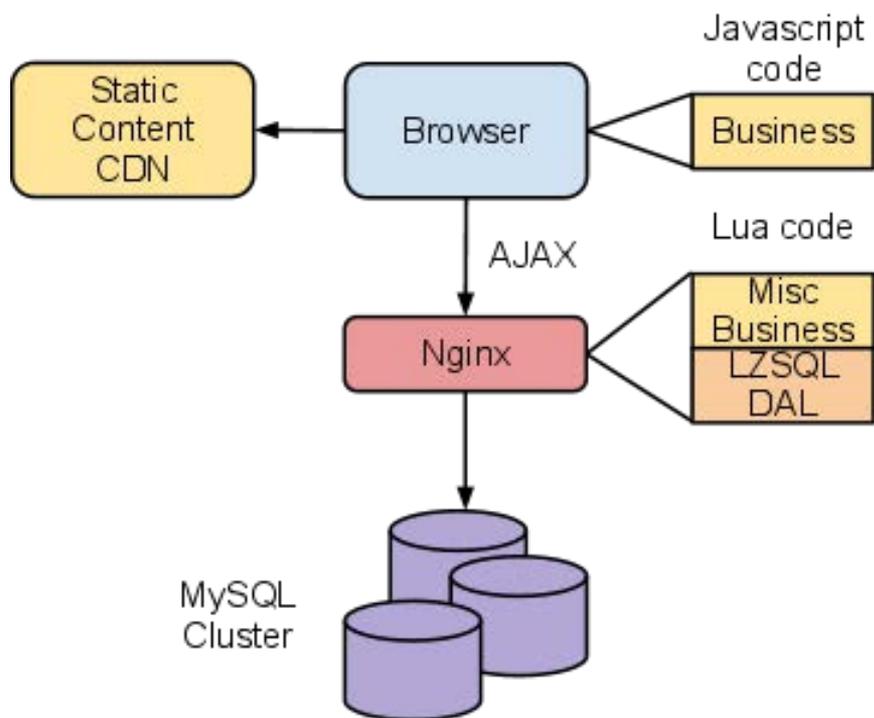


效果

新老接口总体对比 - 平均响应时间 (P4P一期, cache miss)



思考



- 优点
 - 服务侧开发语言同质化，开发效率明显提升
 - 大数据请求处理速度大幅度提高
- 完美的方案？

思考

- 协作式调度模型的问题
- Lua代码死循环
- I/O操作受限于Nginx模型
- 调试功能

参考

- 本演示稿中涉及的大部分技术已经开源：
 - <http://tengine.taobao.org>
 - <https://github.com/chaoslawful/luanginx-module>
 - <https://github.com/chaoslawful/drizzle-nginx-module>
 - https://github.com/agentzh/nginx_openresty

欢迎加入我们!



shudu@taobao.com



新浪微博

[@淘叔度](#)



qingwu@taobao.com



新浪微博

[@chaoslawful](#)