# 手机端API网络性能测试报告

以下是3种网络环境——公司网线，wifi，联通3g网卡，针对线上mobile-api环境做的一个网络性能测试报告。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 功能 | API | 平均响应时间 | | |
| 联通3G网卡 | 公司WIFI | 网线 |
| email登录 | <http://mobile-api.gaopeng-huoban.com/main.action>?  sign=4006e2a8143fe0b7e231c05a569a529c&  appId=GAOPENG\_MOBILE\_APP\_1&phoneType=android  [&email=yuanjuxing@126.com&api=user.login&format=xml&ver=1.0&password=juxing1982](mailto:&email=yuanjuxing@126.com&api=user.login&format=xml&ver=1.0&password=juxing1982) | 135ms | 65ms | 65ms |
| deal详情 | <http://mobile-api.gaopeng-huoban.com/main.action?sign=3fed848d4d663c81d0c1cbc0e3b8fb30>  &appId=GAOPENG\_MOBILE\_APP\_1&phoneType=android&  api=citydeal.detail&format=xml&ver=1.0&dealId=1431736747 | 600ms | 80ms | 75ms |
| deal列表 | <http://mobile-api.gaopeng-huoban.com/main.action>?  sign=6f63384bc42ae34487ea92441fa71d2d&appId=GAOPENG\_MOBILE\_APP\_1  &lastCityName=%E5%8C%97%E4%BA%AC&phoneType=android  &cityName=%E5%8C%97%E4%BA%AC&api=citydeal.lst&format=xml&ver=1.0&offset=0&size=100 | 6.5s | 3.7s | 3.5s |
| 即将过期的优惠券 | <http://mobile-api.gaopeng-huoban.com/main.action?sign=9d4d4ec6fd7192d0123bc5e012b25e46&>  cookie=77070e123e6c242d282e362c182c32263c3b28263967656c677d6e006a707  e7d7576676f7975737176706a61636d3c382c37162a77777d0073727b677578706e7d6d3f790a041d  &appId=GAOPENG\_MOBILE\_APP\_1&phoneType=android&api=couponCloseExpired.lst&format=xml&ver=1.0 | 900ms | 119ms | 100ms |
| 已使用的优惠券订单 | <http://mobile-api.gaopeng-huoban.com/main.action?sign=ae0ab7a1959b7db022974055d8b5cd42&>  cookie=77070e123e6c242d282e362c182c32263c3b28263967656c677d6e006a707e7d  7576676f7975737176706a61636d3c382c37162a77777d0073727b677578706e7d6d3f790a041d  &appId=GAOPENG\_MOBILE\_APP\_1&phoneType=android&api=couponUsed.lst&format=xml&ver=1.0 | 1s | 125ms | 95ms |
| 未使用的优惠券订单 | <http://mobile-api.gaopeng-huoban.com/main.action?sign=a439890c651ff223e0a47abf8140dec3&>  cookie=77070e123e6c242d282e362c182c32263c3b28263967656c677d6e006a707e7d757  6676f7975737176706a61636d3c382c37162a77777d0073727b677578706e7d6d3f790a041d  &appId=GAOPENG\_MOBILE\_APP\_1&phoneType=android&api=couponUnused.lst&format=xml&ver=1.0 | 2.1s | 150ms | 130ms |
| 百度 | [www.baidu.com](http://www.baidu.com/) | 130ms | 45ms | 40ms |
| 糯米 | [www.nuomi.com](http://www.nuomi.com/) | 500ms | 104ms | 100ms |
| 高朋 | [www.gaopeng.com](http://www.gaopeng.com/) | 1.27s | 500ms | 500ms |

从数据结果看，目前通过3G网卡访问手机API的响应时间是公司wifi和网线的3倍以上，而且从百度、糯米、高朋网等情况来看，其响应时间也是在3倍以上。另外，在数据量较大的情况下，响应时间较长，如获取100个deal列表，需要6s以上，但目前deal显示一般不会超过100个，因此通过3G上网卡API在3S内都能返回处理结果。加上手机处理渲染的时间。每次交互还是可以控制在3-5S内完成。3-5s对用户体验影响不会很大。

结论：如果用户使用3G上网卡访问手机端，响应时间的用户体验在可接受范围内。

PS：我们网站的网络性能和糯米网有较大的差距。

补充deal列表取10条数据的情况：

<http://mobile-api.gaopeng-huoban.com/main.action?sign=7d3e9b5a69680b82d99c8ec8a446383b&appId=GAOPENG_MOBILE_APP_1&lastCityName=100&phoneType=android&cityName=北京&api=citydeal.lst&format=xml&ver=1.0&offset=0&size=10>

|  |  |  |
| --- | --- | --- |
| 联通3G网卡 | 公司WIFI | 网线 |
| 1.9s | 840ms | 650ms |

回复

考虑到手机 CPU 的运算能力比电脑要弱，同样是 3G 或者 Wifi ，其实手机的下载速度会更慢

同样渲染显示的速度也慢

在这个时间上 乘以 1.5 是一个比较合适的估算

加速主要考虑 2 方面

(1) 用现有的 CDN 加速手机 API 输出内容 （设置 CDN 针对手机 API 域名的内容不做任何缓存，只是一个简单的代理访问）

(2) 减小手机 API 输出的内容大小

     <1>  API 考虑把一些不需要的内容去除掉，不要输出

     <2>  nginx 配置打开 gzip 针对 text/json  text/xml  application/xml 的压缩， 同时把压缩级别提高到  level 6 ( 缺省应该是 level 2)

2.0 框架设计的时候就支持 API 的性能统计，比如：

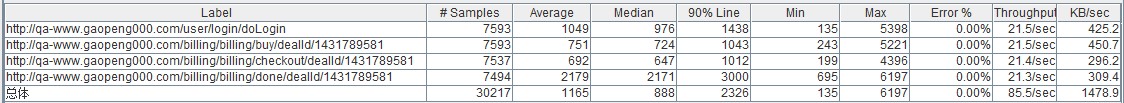
<http://dev-api-mobile.gaopeng000.com/ShowApiPerfServlet>

线上环境只需要打开就可以看到所有的 API 以及对应的每个模块的执行性能

这个报告主要是看一下3G网卡访问线上API的网络状况，判断是否需要将手机的API部署到移动的机房，至于具体API的性能指标永恒和贺凡会用jmeter进行压力测试，并会反馈一份API的性能测试报告。

# 秒杀性能测试结论

相关性能数据如下：



结论：

* 100并发的情况下，支付流程上进行压力测试，系统总吞吐量在85/s左右，数据库服务器cpu idle>70%,内存使用率正常。PHP&java服务器cpu idle>80%.
* 增大并发数会出现ERROR，原因是nigix有最大连接数限制.
* 在当前测试环境下，结合现有网站的pv及峰值。当前性能是可以支撑秒杀的。
* 线上服务器和测试环境服务器性能差距较大，但线上有6台PHP机器，4台JAVA服务器。个人认为应该也可以支撑。但无法评估准确数据。

# frontend2.0性能测试报告

本次整体性能测试对用户登录，deal详情页，deal列表页等页面进行了性能测试，和当前线上版本相比，性能上有非常大的提升，主要体现在两个方面：

1. 从用户体验角度来看，用户打开页面的速度快了很多，在不考虑网络开销以及页面渲染的情况下，旧版frontend请求详情页的时间接近1s，而现在frontend2.0的详情页，在20ms内就能处理完该次请求。对于详情页和列表页，响应时间有接近50倍的提升。大大的提升了用户体验。
2. 从服务器运维成本角度分析，现在线上的机器均是16核的高性能机器，单台机器总的吞吐量在35/sec左右，而frontend2.0的吞吐量在有cache的情况下能够达到500/sec。至少也有10倍的提升。另外，一直被认为是瓶颈的数据库服务器，目前没有任何性能上的压力。由此可以简单的得出，frontend2.0在一样的服务器资源情况下，能够满足更多的业务请求，换句话说，目前PV提升10倍以上，性能也是可以满足的。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 平均响应时间 | 总体吞吐量(单位时间内能处理的请求数) | 备注 |
| Frontend1.0(旧版) | 908ms | 32/sec | Tomcat服务器：16核 |
| Frontend2.0 | 19ms | 512/sec | PHP服务器：8核 |

       API性能测试，对登录，城市列表，deal详情页，高朋券，全部订单，qq联合登录，城市列表等页面发送HTTP请求，为了体现这次API性能优化的显著成果，这里拿手机端API性能做个比较。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 平均响应时间 | 总体吞吐量(单位时间内能处理的请求数) | 备注 |
| Api-frontend | 10ms | 970/sec（同样的16核CPU能上7000/sec） | Tomcat服务器：4核 |
| Api-mobile | 180ms | 380/sec | Tomcat服务器：16核 |

从上述数据看出，frontend2.0的性能较api-mobile有极大的提升，响应时间是提升接近20倍，吞吐量也是接近20倍的提升。API-frontend部分业务也没有涉及cache，由此可以看出，此次重构api的性能优化是非常显著。

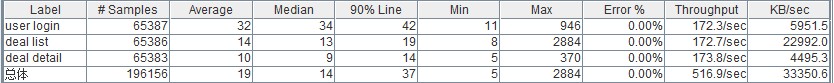
详细性能测试数据如下：

**1.测试场景1**

  场景描述：PHP有cache的情况下，jmeter模拟10个并发用户，分别往登录，城市列表，deal详情页三个页面发送HTTP请求。服务器为8核。

  性能数据

Jmeter性能测试报告如下：



  API服务器性能数据：

Cpu ide：90%左右，CPU消耗掉10%

服务器运行队列：<4

Free memory：运行一段时间后，维持稳定状态

  PHP服务器性能数据：

Cpu ide 55%左右，CPU消耗掉45%

服务器运行队列： 4左右

Free memory：运行一段时间后，维持稳定状态

**性能分析与结论**

**(1)10个线程情况下，API性能理想。**

**(2)PHP的吞吐量和响应时间都比较理想**

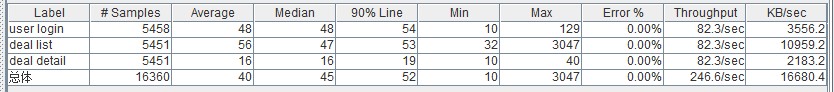
**(3)在发送压力的情况下，直接从页面上访问，用户感觉不到系统变慢。且没有出现异常。**

**2.测试场景1**

  场景描述：PHP无cache的情况下，jmeter模拟10个并发用户，分别往登录，城市列表，deal详情页三个页面发送HTTP请求。服务器为8核。

  性能数据

Jmeter性能测试报告如下：



  API服务器性能数据：

Cpu ide：90%左右，CPU消耗掉10%

服务器运行队列：<4

Free memory：运行一段时间后，维持稳定状态

  PHP服务器性能数据：

Cpu ide 20%左右，CPU消耗掉80%

服务器运行队列： 4左右

Free memory：运行一段时间后，维持稳定状态

**性能分析与结论**

**(1)10个线程情况下，API性能理想。**

**(2)PHP的吞吐量和响应时间都比较理想，CPU消耗较大。但仍然能满足线上要求**

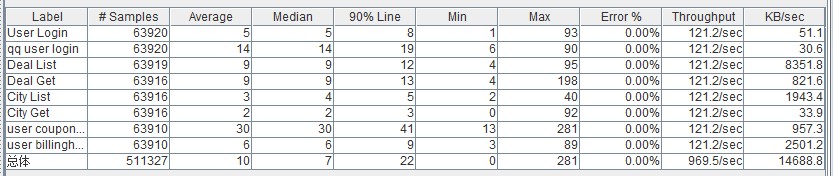
**(3)在发送压力的情况下，直接从页面上访问，用户感觉不到系统变慢。且没有出现异常。**

**3.测试场景1**

  场景描述：API有cache的情况下，jmeter模拟10个并发用户，分别往登录，城市列表，deal详情页，高朋券，全部订单，qq联合登录，城市列表等页面发送HTTP请求。服务器为4核。

  性能数据

Jmeter性能测试报告如下：



  API服务器性能数据：

Cpu ide：20%左右，CPU消耗掉80%

服务器运行队列：>4

Free memory：运行一段时间后，维持稳定状态

**性能分析与结论**

**(1)10个线程情况下，API响应时间及吞吐量均很很好，但CPU消耗较大。**

**(2)在发送压力的情况下，直接从页面上访问，用户感觉不到系统变慢。且没有出现异常。**

谢谢

PHP  CPU 占用这么高 ？  是因为渲染页面的原因吗？ yii 的模板缓存开启试试，应该好些

PHP 把 APC 打开，配置给 128MB 内存， 这样生成的 模板编译后文件 都可以用到缓存，会好些

如果 CPU 占用还是很高，而且还是因为渲染页面，那我建议考虑做 静态化，

最简单的静态化策略，

(1) PHP 调用 API，然后渲染页面，最后把页面结果写入到一个 静态目录下 (目录名和文件名需要好好设计一下，把参数用上)

(2) nginx 配置 URL Rewrite 策略，如果静态页面存在，就直接返回静态页面，如果不存在，就调用相应的 PHP  程序生成

(3) nginx 里面，把这些静态化页面的缓存侧率全部设为  no-cache，不许缓存，每次都重新请求，防止我们刷新了之后用户客户端还是旧的

由于静态化会出现依赖性问题，比如编辑了一个 deal，除了这个 deal 的详情页面需要重新生成，可能对应城市的 deal 页面也要重行生成

所以，最简单的侧率就是，不管这些依赖性，我们每分钟清除一次静态页面

(4) 弄个 cron job ，每分钟把过期页面删除（1分钟前生成的静态页面），这样下次请求就会自动重新生成

每分钟清除一次，就算有不一致的情况，也就 1 分钟，问题不大，但是却可以大大减少 PHP 渲染页面的工作量，

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

或者简单一点，用 Yii 的 文件缓存，把渲染的页面直接缓存(或者在  layout 里面把页面的组件），可以大大降低页面渲染的压力

如果有时间的话，可以用  Xdebug + valgrind 做一下性能剖析 ， 类似 Java 的 jprofiler ，可以看到执行时间到底消耗在哪里

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Java CPU 占用率也蛮高的，说明 memcache 缓存的大多数是 “原始数据”， 这些数据应该都是直接的数据库查询结果吧

所以省去了数据库的查询时间（IO时间），但是后续对这些数据的加工还是要做，所以 CPU 的计算时间没减少，CPU占用很高

要降低 Java 的CPU 占用，可以考虑缓存 “加工后的结果”， 这样一个访问请求过来，直接取这个  “加工后的结果” 返回就可以了，

CPU 占用基本为 0

缓存  “加工后的结果” 会有一个问题，就是  依赖性的缓存 更新，一个 deal 更新了，对应的 deal 详情、deal列表、... 一串都需要更新

需要建立一个依赖链 (dependency chain)，对这个链上的数据都更新缓存，这个需要好好设计和考虑一下，会有难度

# 手机API

我对手机API做了下简单的压力测试（我本机环境）

OS: VM ubuntu10.04

Mem: 2G

Threads: 100

Ramp-up Period: 1s

Start Time: 2011/11/09 14:23

End Time: 2011/11/09 14:53

Duration: 30min

附件中是测试结果。

赞啊，不过建议在QA或一台性能较好的机器上在重新压个1H左右看看结果。因为从目前测试结果看，deal list,deal detail，common city list的平均响应时间在10S左右。其他的API平均响应时间都在3S以内。deal list,deal detail，common city list的响应时间有些超长，估计和机器性能有很大的关系，因此建议在接近线上机器性能的情况下评估一个较准确的值。

# api-mobile性能测试结论

Hi all：

         以下是相关性能测试结论与数据：

**性能测试结论**

在并发压力100的情况下，系统平均响应时间为180ms，总体吞吐率达到每秒500请求以上，且服务器各项性能指标稳定。从这这几项数据得出：API性能良好，能够承受较大并发压力，按照目前网站端200万PV计算，每秒并发处理请求数在30左右，而现在系统吞吐率在500以上，能承受现有10倍的PV；也就是说，手机端每日PV上2000万，目前api-mobile的性能也是可以支持的。因此，性能测试通过。

具体场景和详细数据请查看以下内容：

**场景一**

**【场景描述】**

服务器连接池配置30，混合压力测试，共涉及API有：Deal List、Deal Detail、Common City List、Characteristic City List、Get Qq User、Get Close To Expired Coupon List、Get Already Used Coupon List、Get Unused Coupon List、Get Entity List

1. 每秒并发100个请求，压力通过apache转发给tomcat

**【主要性能数据及其分析】**

**【具体数据】**

**Jemter报告：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| API接口 | 请求数 | 平均响应时间 | 最小响应时间 | 最大响应时间 | 错误率 | 吞吐率(每秒处理请求数) |
| Deal List | 8718 | 282 | 23 | 55921 | 0 | 42 |
| Deal Detail | 8716 | 177 | 14 | 115448 | 0 | 43 |
| Common City List | 8703 | 147 | 25 | 58711 | 0 | 43 |
| Characteristic City List | 8694 | 98 | 11 | 112242 | 0 | 43 |
| Get Qq User | 8684 | 129 | 11 | 56238 | 0.010018425 | 43 |
| Get Close To Expired Coupon List | 8672 | 205 | 11 | 126588 | 0 | 43 |
| Get Already Used Coupon List | 8658 | 131 | 11 | 108969 | 0 | 43 |
| Get Unused Coupon List | 8640 | 258 | 11 | 110482 | 0 | 43 |
| Get Entity List | 8629 | 168 | 11 | 110638 | 0 | 43 |
| 总体 | 78114 | 177 | 11 | 126588 | 0.001113757 | 384 |

**服务器相关性能数据:**

Cpu ide：基本在70%以上

服务器运行队列：>6

Free memory：运行一段时间后，维持稳定状态

**【性能结论】**

1. 系统每个请求的平均响应时间为177ms，总体吞吐率达到每秒380请求。从这两项数据得出：服务器性能良好，能够承受较大并发压力，按照目前网站端200万PV计算，每秒处理请求数在30左右，而现在系统吞吐率在300以上，能承受现有10倍的PV；也就是说，手机端每日PV上2000万，目前api-mobile的性能也是可以支持的。
2. 服务器cpu.idle, vmsize, handle各项指标符合预期，性能在测试一段时间后，系统各项指标均很多稳定；
3. Get Qq User有一定错误率，是由于数据库中有脏数据导致的，从这种情况可以得出结论，在系统出现500错误时，系统服务都能正常运转。没有出现连接不释放等情况。

**场景二**

**【场景描述】**

服务器连接池配置30，混合压力测试，共涉及API有：Deal List、Deal Detail、Common City List、Characteristic City List、Get Qq User、Get Close To Expired Coupon List、Get Already Used Coupon List、Get Unused Coupon List、Get Entity List

1. 每秒并发100个请求，压力直接发给tomcat

**【主要性能数据及其分析】**

**【具体数据】**

**Jemter报告：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| API接口 | 请求数 | 平均响应时间 | 最小响应时间 | 最大响应时间 | 错误率 | 吞吐率(每秒处理请求数) |
| Deal List | 39832 | 404 | 20 | 21899 | 0 | 60 |
| Deal Detail | 39807 | 199 | 11 | 12142 | 0 | 61 |
| Common City List | 39799 | 198 | 12 | 21619 | 0 | 61 |
| Characteristic City List | 39793 | 77 | 5 | 9105 | 0 | 61 |
| Get Qq User | 39791 | 81 | 5 | 11923 | 0.010002 | 61 |
| Get Close To Expired Coupon List | 39790 | 110 | 5 | 9349 | 0 | 61 |
| Get Already Used Coupon List | 39789 | 119 | 4 | 9581 | 0 | 61 |
| Get Unused Coupon List | 39770 | 283 | 5 | 21442 | 0 | 61 |
| Get Entity List | 39766 | 160 | 5 | 22689 | 0 | 61 |
| 总体 | 358137 | 181 | 4 | 22689 | 0.001111 | 547 |

**服务器相关性能数据:**

Cpu ide：基本在70%以上

服务器运行队列：>6

Free memory：运行一段时间后，维持稳定状态

**【性能结论】**

1. 在直连tomcat的情况下，系统的单位时间内所能处理的请求数较tomcat转发的情况下有较大幅的提升。

非常感谢永恒的辛苦优化，也是希望能够将这些经验总结分享出来，供后续2.0重构借鉴。

谢谢

通过这次api-mobile性能测试，总结一下：

1. 一开始压测发现一段时间后请求发送变慢，内存占用很大，通过巨星和伍星帮助排查是查询订单记录性能较差。 因为查询一个订单时，如果是child deal需要查询一次他的父deal，用父deal来判断是优惠券还是实体订单（因为数据库中这个数据的判断只能根据父deal判断）；之后还要查询一次当前deal，查出一些相关显示的数据。由于订单过多的话，循环查询数据库次数过多，导致内存中存放的数据过大，导致垃圾回收压力很大。  确定原因后，做了修改，连表查出相关deal的信息，同时根据child deal来判断订单是优惠券还是实体订单（需要数据库同步parent deal和child deal postagepayment字段的值）。 修改后性能提升了10倍以上，且服务器各项指数正常。
2. 伍星帮忙调整下tomcat的jvm参数等，初始化分配的堆内存过小
3. 发现不通过apache直接通过tomcat的压测的性能比通过apache的测试结果快10倍，需要调整一下apache的配置。

总结数据库的一些表结构合理性对代码性能影响很大，应用服务器的配置参数以及jvm的参数需要根据要求的性能指标提前做一些测试。

         还发现一些数据库合理性的问题

1. users\_joined\_deal 的shipAddr(255) = ua\_mobile+ua\_receiver\_name+ua\_cityname+ua\_address+ua\_postcode但是 users\_address中 ua\_mobile (16), ua\_receiver\_name (64), ua\_cityname (255), ua\_address (255),  ua\_postcode (16) 总长度>255 ---- 存在风险。
2. users 表中的qq\_connect\_id存在大量不唯一的记录。