6

第 6 章

性能测试实战

通过前面的章节，我们对如何使用 LoadRunner 工具进行性能测试的原理和方法有了深入的了解，但性能测试不是 LoadRunner，作为一个优秀的性能测试工具，掌握了它并不代表你已经会做性能测试。水能载舟亦能覆舟，如果只将性能测试定位在 LoadRunner 这个工具所提供的方案上，是无法进行有效的性能测试的。

性能测试是一个综合性的工作，在前面的工具介绍中，我们也不断提到性能测试的基础是需求，脚本的开发需要被测系统开发人员的配合，而性能瓶颈定位需要各个部门的通力协作。如何在工作中进行性能测试？如何获得性能需求？性能测试需要编写哪些文档？性能测试结果如何进行分析？如何成立性能测试部门？如何对性能测试进行自动化？对于以上问题你将会在本章的学习中一一得到解答。

出于实践方便的考虑，我们选择了 Discuz!.NET 论坛——一款知名的免费开源论坛系统。我们将对其进行一次完整的性能测试，来了解实施性能测试的流程和思路。我们将对 Discuz!2.1 和 2.5 进行性能基准对比测试，对 Discuz!2.5 进行负载测试。

6.1 性能测试流程

无规矩不成方圆，在执行性能测试之前，我们先来了解一下常见的性能测试流程，如图 6.1 所示。

6.1.1 计划测试

在任何类型的测试中，编写测试计划都是必要的步骤。有条不紊、计划周密的计划，可以确保在执行中能够有章可循。在计划测试阶段需要输出性能测试计划，而计划阶段需要经历以下几个环节，如图 6.2 所示。

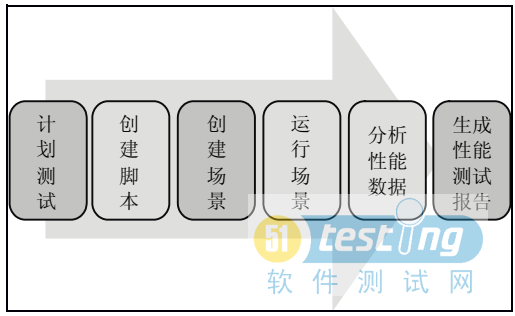


图 6.1　性能测试流程

分析系统阶段

要进行性能测试，了解被测对象是需要做的第一步。首先需要确认系统的架构和所使用的协议，对工具的可行性进行分析，然后对整个业务进行熟悉，确认相关的数据和业务操作可以被工具录制回放。

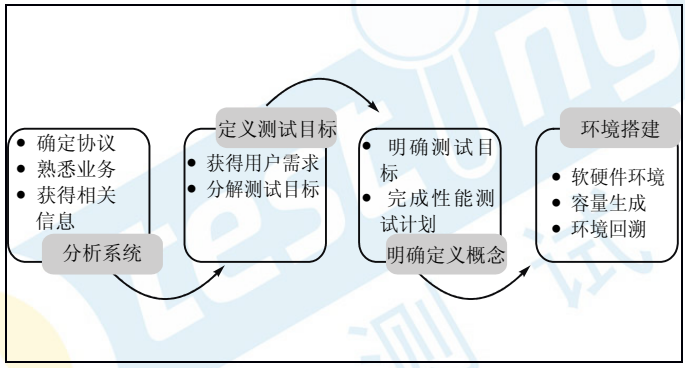


图 6.2　性能测试计划阶段流程

✓　通过分析系统阶段需要知道该系统能不能进行性能测试。

确定协议

……此处内容略

✓　通过定义测试目标需要知道的是用户想要什么。

那么如何获得性能需求呢？在此之前请先思考一个问题：上海地铁人民广场站 1 号线站台和换乘大厅的楼梯应该修建多宽？

乍眼一看，你可能觉得这是一个功能需求，怎么和性能有关系呢？但这个确实是性能问题，如果换乘的楼梯修建过窄，会导致大量乘客无法疏散出月台引起事故，而如果修建过宽又会带来不必要的浪费，合适的疏散能力就是性能的吞吐量指标。

可能有些朋友会说，我没去过上海，也没坐过地铁，如何知道这个性能是多少呢？确实，很多时候我们对用户所需要的系统闻所未闻，现在需要我们来做性能需求定义性能测试的目标不是无稽之谈么？那么按照这个道理来说，做性能需求分析的人都应该是业务专家，从某些角度来说需求分析本来就是需要业务或技术的专家才能担当，但并不是说没有简便入手的方法，对于这种情况，可以通过以下几个方法来获得系统的性能需求。

通过用户提供的数据进行分析

用户永远是需求的最后确认人，通过用户介绍和提供的相关数据是分析系统性能需求最直接也是最简便的方式。例如用户想要新建一个系统，来将公司的业务无纸化，那么他会提出相关的要求，例如容量、响应时间、并发量、服务器资源等。而作为实现方，当性能需求分析工程师将用户的需求进行整理后，需要参考以前的项目或相关信息去确定该需求是否合理可实施，进一步和用户协调制定合理有效的性能需求，以达到双赢的结果。

通过系统日志

当进行旧系统升级的时候，历史数据即系统日志的方式是获得真实用户需求最有效的参考数据。作为常用的 Web 服务器 IIS 或者 Apache，当用户通过客户端发送请求到 Web 服务器的时候，IIS 或者 Apache 都会在访问日志中记录下来。

IIS 日志设置的方法说明如下。在 IIS 中找到需要的网站，打开日志功能，如图 6.14 所示。

图 6.14　IIS 7.0 日志设置

IIS 默认支持以下 4 种日志格式：

- Microsoft IIS 日志文件格式
  - 信息记录在以逗号分隔的 ASCII 文本文件中。
  - 数据是固定的，不能自定义该日志。
  - 单个传输有多条记录。
- NCSA 公用日志文件格式
  - 信息记录在使用（美国）国家超级计算技术应用中心（NCSA）格式的 ASCII 文本文件中。
  - 数据是固定的，不能自定义该日志。
  - 单个传输有多条记录。
- W3C 扩展日志文件格式

- W3C（万维网联合会）扩展日志文件格式是 SMTP 服务以及其他 IIS 服务的默认日志记录格式。
- 数据是变化的，可以选择要跟踪的内容。
- 此种格式是限制日志大小很好的选择。
- 信息记录在 ASCII 文本文件中。

图 6.15　IIS 7.0 W3C 日志属性设置

- 这种格式包括某些仅适用于 Web 和文件传输协议（FTP）服务的字段选项。
- 单个传输有多条记录。

- ODBC 日志记录

- 使用此格式前，必须建立一个与开放式数据库连接（ODBC）兼容的数据库。
- 信息记录在数据库中。
- 单个传输有多条记录。

如果使用 IIS 作为 Web 服务器的话，建议使用 W3C 格式。这里可以通过格式后的选择字段功能为日志添加监控属性，如图 6.15 所示。

监控的数据越多，对后期分析可以提供的支持就越强。

Apache 日志设置的方法说明如下。Apache 的访问日志格式支持自定义，打开 Apache 下的 httpd.conf 文件，使用 LogFormat 命令来查找系统默认的日志格式，也可以自行定义需要的日志格式。

在 httpd.conf 文件中找到以下内容：

```
    <IfModule log_config_module>
        #
        # The following directives define some format nicknames for use with
        # a CustomLog directive (see below).
        #
        LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
        LogFormat "%h %l %u %t \"%r\" %>s %b" common

        <IfModule logio_module>
          # You need to enable mod_logio.c to use %I and %O
          LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"
%I %O" combinedio
        </IfModule>

        #
        # The location and format of the access logfile (Common Logfile Format).
        # If you do not define any access logfiles within a <VirtualHost>
        # container, they will be logged here.  Contrariwise, if you *do*
        # define per-<VirtualHost> access logfiles, transactions will be
        # logged therein and *not* in this file.
        #
```

```
    CustomLog "c:/logs/access.log" common

    #
    # If you prefer a logfile with access, agent, and referer information
    # (Combined Logfile Format) you can use the following directive.
    #
    #CustomLog "logs/access.log" combined
</IfModule>
```

我们先来简单介绍一下 LogFormat 的使用格式。以下是一个典型的记录格式：

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

它定义了一种特定的记录格式字符串，并给它起了个别名叫 common，其中的"%"指示服务器用某种信息替换，其他字符则不作替换。引号（"）必须加反斜杠转义，以避免被解释为字符串的结束。格式字符串还可以包含特殊的控制符，如换行符"\n" 、制表符"\t"。

CustomLog 指令建立一个使用指定别名的新日志文件，除非其文件名是以斜杠开头的绝对路径，否则其路径就是相对于 ServerRoot 的相对路径。

上述配置是一种被称为通用日志格式（CLF）的记录格式，它被许多不同的 Web 服务器所采用，并被许多日志分析程序识别，它产生的记录形如：

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0"
200 2326
```

记录的各部分说明如下。

```
127.0.0.1（%h）
```

这是发送请求到服务器的客户 IP 地址。如果 HostnameLookups 设为 On ，则服务器会尝试解析这个 IP 地址的主机名并替换此处的 IP 地址，但并不推荐这样做，因为它会显著拖慢服务器，最好是用一个日志后续处理器来判断主机名，比如 logresolve。如果客户和服务器之间存在代理，那么记录中的这个 IP 地址就是代理的 IP 地址，而不是客户机的真实 IP 地址。

```
-（%l）
```

这是由客户端 identd 进程判断的 RFC1413 身份（identity），输出中的符号“-”表示此处的信息无效。除非在严格控制的内部网络中，此信息通常很不可靠，不应该被使用。只有在将 IdentityCheck 指令设为 On 时，Apache 才会试图得到这项信息。

```
frank（%u）
```

这是 HTTP 认证系统得到的访问该网页的客户标识（userid），环境变量 REMOTE_USER 会被设为该值并提供给 CGI 脚本。如果状态码是 401，表示客户未通过认证，则此值没有意义。如果网页没有设置密码保护，则此项将是“-”。

```
[10/Oct/2000:13:55:36 -0700] (%t)
```

这是服务器完成请求处理时的时间，其格式是：

```
[日/月/年:时:分:秒 时区]
日=2 个数字位
```

```
月=3 个字母位
年=4 个数字位
时=2 个数字位
分=2 个数字位
秒=2 个数字位
时区 = (+正|-负)4 个数字位
```

可以在格式字符串中使用 %{format}t 来改变时间的输出形式，其中的 format 与 C 标准库中的 strftime()用法相同。

```
"GET /apache_pb.gif HTTP/1.0" (\"%r\")
```

引号中是客户端发出的包含许多有用信息的请求行。可以看出，该客户的动作是 GET，请求的资源是/apache_pb.gif，使用的协议是 HTTP/1.0。另外，还可以记录其他信息，如：格式字符串"%m %U%q %H"会记录动作、路径、查询字符串、协议，其输出和"%r"一样。

```
200 (%>s)
```

这是服务器返回给客户端的状态码。这个信息非常有价值，因为它指示了请求的结果，或者是被成功响应了（以 2 开头），或者被重定向了（以 3 开头），或者出错了（以 4 开头），或者产生了服务器端错误（以 5 开头）。完整的状态码列表参见 HTTP 规范（RFC2616 第 10 章）。

```
2326 (%b)
```

最后这项是返回给客户端的不包括响应头的字节数。如果没有信息返回，则此项应该是“-”，如果希望记录为“0”的形式，就应该用%B。

首先使用 LogFormat 命令 LogFormat "%h %l %u %t \"%r\" %>s %b" common 定义日志格式，然后再通过 CustomLog 命令 CustomLog "c:/logs/access.log" common 为日志设置存放的地址。这样就完成了 Apache 日志的设定工作。

接着让用户在正常情况下使用一段时间这个配置了日志的系统，就能得到大量的访问数据，以便于进行下一步的日志分析工作。

什么是日志分析？虽然现在得到了用户的访问日志，但是面对数以亿计的用户请求记录，如何才能从这众多的数据中找到想要的数据，这个时候就需要通过日志分析工具来实现了，这也是为什么在前面需要规范日志格式的原因，否则工具会不支持自定义的日志格式。

日志的分析工具其实也很多，这里介绍的是 WebLog Expert。作为一个强有力的日志分析工具，WebLog Expert 能够分析网站的流量记录，将原始的流量记录分析出 Activity statistics、Access statistics、Information about visitors、Referrers、Information about errors 等基本而重要的流量信息，帮助你了解网站的使用状况，另外它同时支持 IIS 和 Apache 日志也是我们选用该软件的原因之一。

整个工具的使用比较简单。打开 WebLog Expert 单击新建按钮，创建一个新的分析日志项目，填写项目名称和服务器地址后，选择事先准备好的的日志文件，如图 6.16 所示。

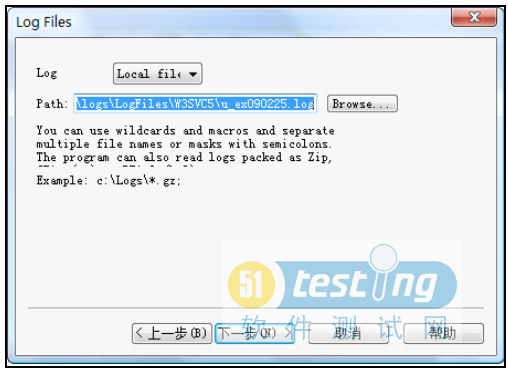


图 6.16 WebLog Expert 日志文件设置

一般来说 IIS 的日志都默认存放在 inetpub\logs\LogFiles 地址下，在这个目录下会看到以 W3SVC1 开头的目录，如果有多个网站在一个 IIS 上那么目录会依次按照序号排下去，找到需要分析的网站日志目录，该目录中有很多.log 文件，这些文件就是用户访问 IIS 所留下的访问日志。如果是 Apache 日志请检查 Httpd.conf 文件中定义的文件存放地址。

添加完成后，单击下一步按钮，确认需要分析的时间段，如图 6.17 所示。

Weblog Expert 支持多种时间过滤规则，由于这里实例的日志比较小，所以我们选择 All activity 选项对日志中所有的时间段进行分析，后面的跟踪类型和过滤规则一般不设置，接着转到最关键的地方，如图 6.18 所示。

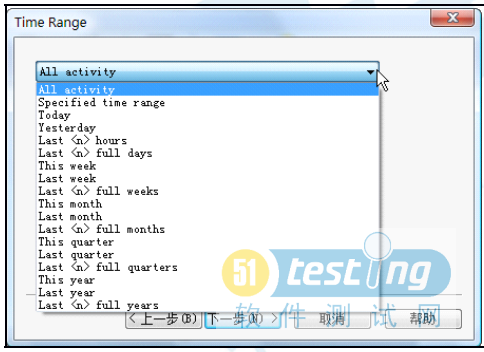


图 6.17 WebLog Expert 日志分析时间设置

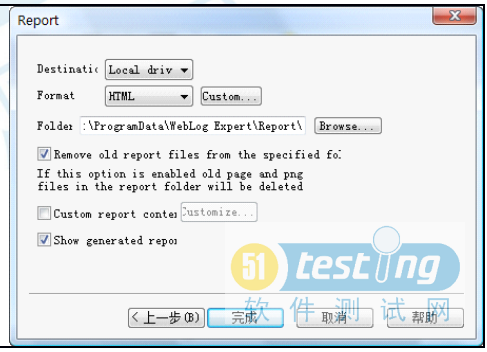
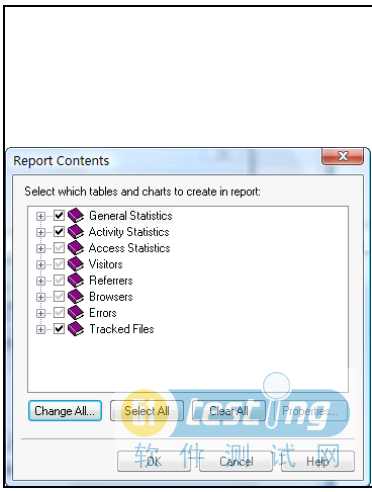


图 6.18 WebLog Expert 日志生成格式设置

使用日志分析工具的核心就是生成的报告能帮助我们挖掘到那些感兴趣的内容，这里 WebLog Expert 支持输出 HTML/PDF/CVS 等格式的分析报告，还可以自定义输出的格式和分析报告的内容，选中 Custom report content 后可以自定义需要输出的报告内容，如图 6.19 所示。

系统提供了大量的报告选项可以根据情况进行定义。当这些都做完后，单击完成按钮，结束日志分析设置。单击菜单中的 Analyze 按钮，稍等片刻便会生成一份详尽的日志分析报告（用户行为记录），如图 6.20 所示。



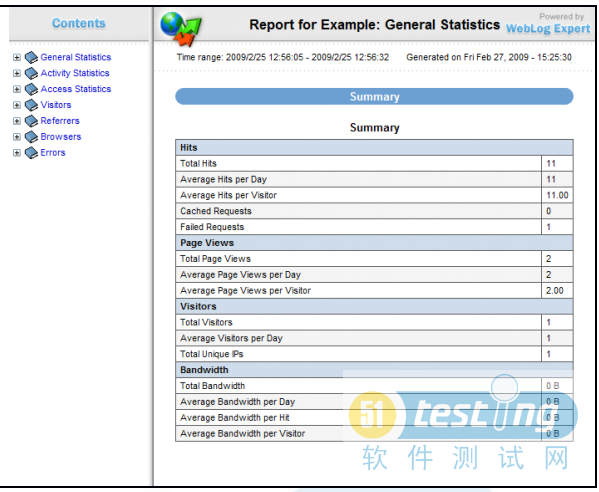


图 6.19　WebLog Expert 自定义生成日志的内容　　图 6.20　WebLog Expert 生成日志分析报告

通过这份报告，可以得到系统当前的负载情况，如果能够结合同时期的系统资源日志和系统错误日志，就能够得到非常珍贵的性能测试数据（系统在过去的负载情况以及对应资源的利用率和响应时间），为后期的性能调优提供基准数据。接着来看看一个真实的系统所得到的日志数据，如表 6.1 所示。

表 6.1　系统数据 Summary

Time range: 2008/8/19 22:15:24 - 2008/9/1 20:51:53　　Generated on Mon Sep 22, 2008 - 11:59:35

Summary

Hits	
Total Hits	29,930,396
Average Hits per Day	2,137,885
Average Hits per Visitor	51.48
Cached Requests	13,244,961
Failed Requests	754,898
Page Views	
Total Page Views	6,340,382
Average Page Views per Day	452,884
Average Page Views per Visitor	10.90
Visitors	
Total Visitors	581,427
Average Visitors per Day	41,530
Total Unique Ips	315,632
Bandwidth	
Total Bandwidth	1849.27 GB
Average Bandwidth per Day	132.09 GB

Average Bandwidth per Hit	64.79 KB
Average Bandwidth per Visitor	3.26 MB

这是某个论坛系统在 2008 年 8 月 19 日至 2008 年 9 月 1 日的系统访问日志分析报告，整个日志大约有 12GB，分析这个日志文件也需要点时间。

通过这个简单的 Summary 可以粗略地了解系统当前的一些基础性能指标。在大约两周的时间内，整个系统共有 581427 人访问，共产生 29930396 次点击，总带宽使用 1849.27GB。

是不是得到这些数据就能做性能测试了呢？非也，平均数据和总计数据其实对于做性能测试并没有什么用处，因为访问并不是平均的，从测试的角度，更在意峰值数据，整个系统只要能够满足用户在真实访问下的峰值数据，就能保证整个系统能够满足用户的性能需求。所以一般桥或堤坝都是按照 50 年一遇标准或者 100 年一遇标准来修建，其实就是根据历史记录获得峰值情况，按照这个标准来衡量性能。

那么如何获得峰值数据呢？我们来看看在这份日志报告中还能得到什么。首先如果想知道在这段时间内，哪一天的访问量是最高的，可以找到日志中的 Activity Statistics 标签，这里提供的都是访问量的统计，打开该标签即可得到在这段时间内系统的访问数据。首先映入眼帘的是每日访问量（如图 6.21 所示）、每日点击量（如图 6.22 所示）、每日带宽统计表（如图 6.23 所示）和生成这些图所使用的数据表（如表 6.2 所示）。

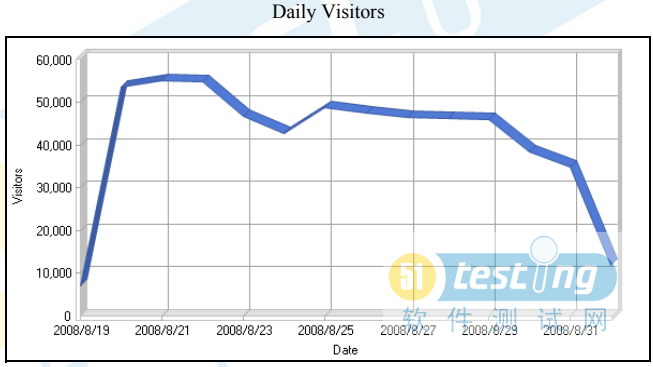


图 6.21　每日访问量统计

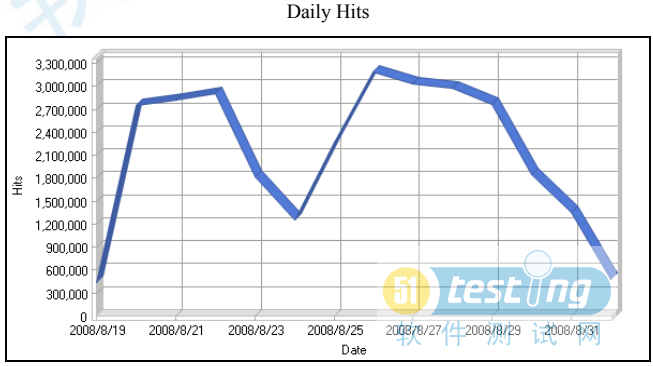


图 6.22　每日点击量统计

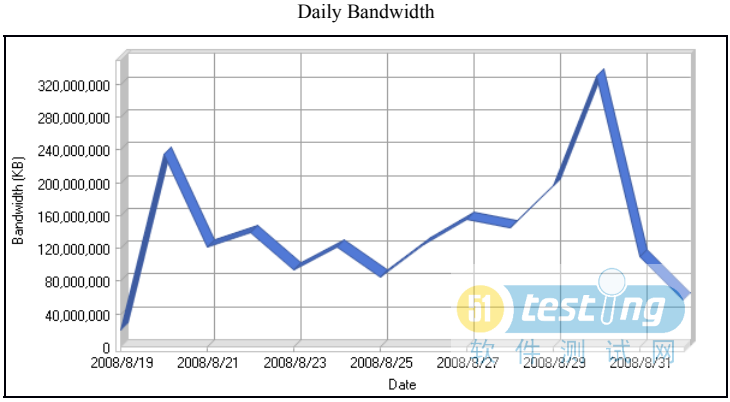


图 6.23　每日带宽统计

表 6.2　每日数据分析表

Date	Hits	Page Views	Visitors	Average Visit Length	Bandwidth (KB)
Tue 2008/8/19	434,414	52,745	7,006	35:14	20,344,842
Wed 2008/8/20	2,748,087	518,739	53,738	14:49	234,160,955
Thu 2008/8/21	2,823,361	521,378	55,263	14:36	122,409,440
Fri 2008/8/22	2,913,660	651,944	54,998	15:56	140,089,499
Sat 2008/8/23	1,817,904	532,110	46,947	14:30	94,513,643
Sun 2008/8/24	1,261,078	346,792	42,741	15:32	121,878,834
Mon 2008/8/25	2,274,538	482,017	48,886	16:43	84,412,575
Tue 2008/8/26	3,185,667	586,778	47,500	17:53	125,310,525
Wed 2008/8/27	3,032,705	558,642	46,648	16:57	156,203,078
Thu 2008/8/28	2,977,549	540,504	46,360	19:25	144,683,988
Fri 2008/8/29	2,772,113	535,783	46,042	18:45	196,891,020
Sat 2008/8/30	1,853,726	458,623	38,499	19:35	330,775,943
Sun 2008/8/31	1,351,513	424,840	34,931	20:07	110,345,823
Mon 2008/9/1	484,081	129,487	11,868	11:16	57,084,709
Total	29,930,396	6,340,382	581,427	17:00	1,939,104,880

通过这些数据，可以轻松地获得系统的每日峰值吞吐量和峰值访问量，从而建立性能测试需求的指标（目标场景的依据），另一方面也得到了用户访问系统的真实情况（手工场景），那么是不是可以参考这个数据来生成场景了呢？

无论对于目标场景还是手工场景来说，上面的这些数据以天为单位还是大了一些，无法得到更精确的数据，不利于性能分析。那么继续往下看日志分析，即可得到每天各小时访问量统计图，如图 6.24 所示。

当看到表 6.3 的时候，大家肯定有一种豁然开朗的感觉，原来想得到的信息就是这个啊。在表 6.2 中可以看到一天中各个时间点上的系统访问量，在这个周期中可以发现对于每天来说用户的访问量并不是平均的。

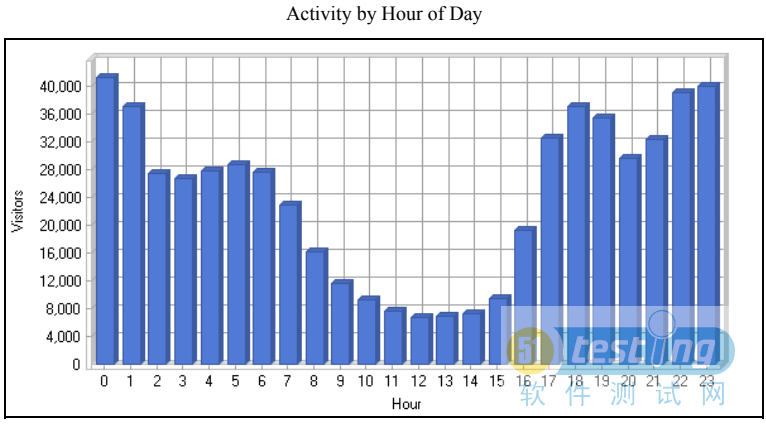


图 6.24　按照时间分布的访问量

表 6.3　小时数据分析表

Hour	Hits	Page Views	Visitors	Bandwidth (KB)
00:00 - 00:59	2,679,676	399,673	41,440	159,594,667
01:00 - 01:59	2,394,003	379,653	37,185	92,646,871
02:00 - 02:59	1,228,717	277,407	27,458	105,742,302
03:00 - 03:59	1,022,261	248,348	26,774	69,052,083
04:00 - 04:59	1,111,897	258,010	27,934	128,603,464
05:00 - 05:59	1,211,130	268,786	28,849	78,760,903
06:00 - 06:59	1,188,914	254,644	27,669	76,709,966
07:00 - 07:59	978,139	233,761	22,922	58,312,606
08:00 - 08:59	615,307	215,040	16,228	52,513,378
09:00 - 09:59	404,471	190,886	11,662	48,162,699
10:00 - 10:59	298,058	178,247	9,217	49,749,937
11:00 - 11:59	254,528	182,380	7,675	50,344,020
12:00 - 12:59	226,321	173,343	6,753	39,264,223
13:00 - 13:59	211,473	171,143	6,870	23,990,368
14:00 - 14:59	231,740	174,559	7,220	15,448,635
15:00 - 15:59	297,573	171,962	9,495	22,370,336
16:00 - 16:59	775,510	213,493	19,385	52,135,771
17:00 - 17:59	1,933,835	313,436	32,659	86,120,656
18:00 - 18:59	2,370,620	359,315	37,130	84,751,826
19:00 - 19:59	2,286,487	353,469	35,483	127,332,111
20:00 - 20:59	1,410,984	271,474	29,752	105,298,515
21:00 - 21:59	1,799,577	304,914	32,467	113,020,056
22:00 - 22:59	2,424,844	361,814	39,111	137,908,514
23:00 - 23:59	2,574,331	384,625	40,089	161,270,961
Total	29,930,396	6,340,382	581,427	1,939,104,880

思考一个问题，像新浪这样的门户网站，每天的访问趋势是什么样的？

图 6.25 是预估的新浪用户访问趋势图，虽然不能拿到真实的访问统计，但是这个数据

有一定的仿真度。为什么这么说呢？大家思考一下自己的访问习惯自然就会找到答案：

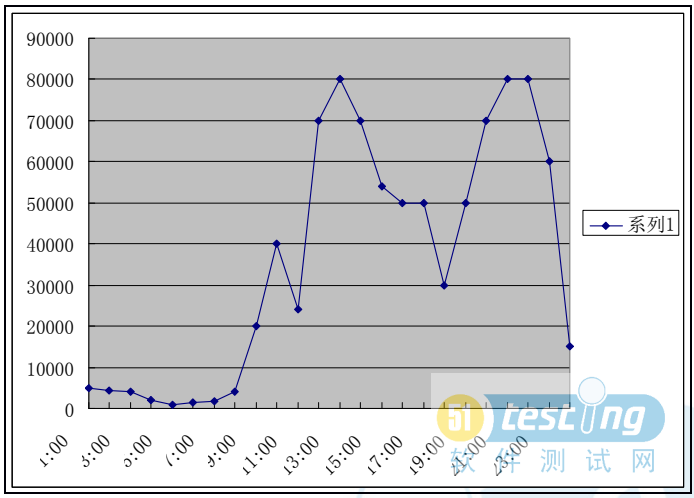


图 6.25　按照时间分布的新浪预估访问量

1．睡觉的时间不会访问（0：00～6：00）。

2．7：00 以后某些朋友会看一下新闻，或者上班时间比较早的人会开始查看自己感兴趣的内容。

3．大多数公司都是在 8：30 以后上班，大家上班的第一件事就是打开电脑，收电子邮件，顺便泡个咖啡，看看新闻，甚至“停停车”……

4．10：00 左右要干活了，但是访问量并不会明显下降，因为不同公司上班的时间不同，晚一点上班的人会将访问量保持在一个层次。

5．12：00 以后是午休时间。一边吃着午饭，一边看看财经新闻和体育新闻是很多朋友的习惯。

6．到了下午 2：00 基本上都开始专心干活了，访问量开始减少，而在下班时间，访问量又会有一个小的反弹，然后继续萎缩，大家都要回家了。

7．晚上吃过晚饭后，迎来最后一个访问提升。

所以基本上每类网站都遵守一个特定的访问量规律，这是由访问者的用户特征决定的。

现在回到当前系统的业务分析中，通过日志分析可以发现一天的访问量是出现在图上的 0：00（注意日志时间是根据该操作系统的系统时间设定的，按照时差计算转换为中国时间需要倒扣 8 小时），也就是北京时间 PM4：00，这就是系统的峰值数据，在这一个小时中受到的负载量如表 6.4 所示。

表 6.4　峰值访问量表

Hour	Hits	Page Views	Visitors	Bandwidth (KB)
00:00 - 00:59	2,679,676	399,673	41,440	159,594,667

1 个小时内产生了 260 万的点击量、接近 40 万的页面刷新、4.1 万访问用户和 159GB 的带宽。这里可以计算一下系统有没有出现带宽瓶颈，如果系统是在机房托管，带宽为 1000Mbps 那么换算一下就是每秒提供 120MB 的带宽，换算成小时是 432000MB 的带宽也

就是 432GB/小时，所以还没有达到带宽峰值，没有明显的带宽瓶颈出现。那么什么时候会出现带宽瓶颈呢？

当系统的访问量达到 600 万点击量、120 万页面刷新、12 万在线用户的时候，带宽已经无法承受这些负载带来的带宽需求了，结果就是每个用户所能分享到的带宽降低，访问速度变慢。

通过以上数据可以得到最终的峰值性能需求（如果需要精确到分钟的峰值性能，那么该工具无法提供，可以自行编写程序对日志进行分析），另一方面场景设置中的手工场景已经可以设计了，只需要按照该日志的走势就可以得到场景模型生成 Real-Life Scenario。

目标场景设计（预估系统扩张规模，预留 20%增长空间）

在一小时内，系统承受 5 万用户在线、页面刷新 48 万次、点击量 320 万、带宽吞吐量 191GB，在该负载下 CPU 及内存资源利用率低于 80%，无明显磁盘网络瓶颈。

手工场景（Real-life Scenario）

根据日志中的用户访问趋势，可以设计得到如图 6.26 所示的 Real-life Scenario（计算了 20%的富余）。

图 6.26 Real-life Scenario

手工场景（Classic）

通过日志得知了最大用户在线数，就可以得到 Classic 下的手工场景。设置用户最大访问量为 5 万用户，持续 1 小时，根据具体情况（设置负载生成速度和负载取消速度），了解在负载逐渐增加到 5 万用户的过程中，系统各资源是否逐渐走向瓶颈，在峰值负载下，获得系统的峰值性能数据，并验证能否满足用户需求。

除了目标场景和手工场景的信息以外还能获得什么信息呢？对于性能测试来说，其实并不需要整个系统的所有功能都达到某个性能，因为有些功能用得多，那么性能要求就高，而某些功能的使用率非常低，性能方面自然就不做要求。通过日志可以得到哪些功能和页面被经常访问，以确定整个系统承受负载的功能集中在哪里，性能测试脚本的目的性会更强。

打开日志的 Access Statistics 功能，即可得到整个系统用户访问最多的页面分析，还有系统中的文件、图片、进入系统页面、用户在系统中的操作路径等信息。通过这些信息可以更加准确地编写有效的测试脚本来模拟大多数用户行为，确保性能负载的真实性。后期的页面优化也可以参考这里的常用访问记录。

在 Daily Page Access 图中可以找到系统被访问最多的页面（业务），如图 6.27 所示。

Visitors 表示访问系统的用户信息，对于系统来说还需要考虑访问用户的所属地，由于

各地的网络发达程度不同，并且还有跨国访问的问题，那么这里还要考虑模拟用户的地区性。通过模拟带宽的变化，来模拟用户访问系统的感受。

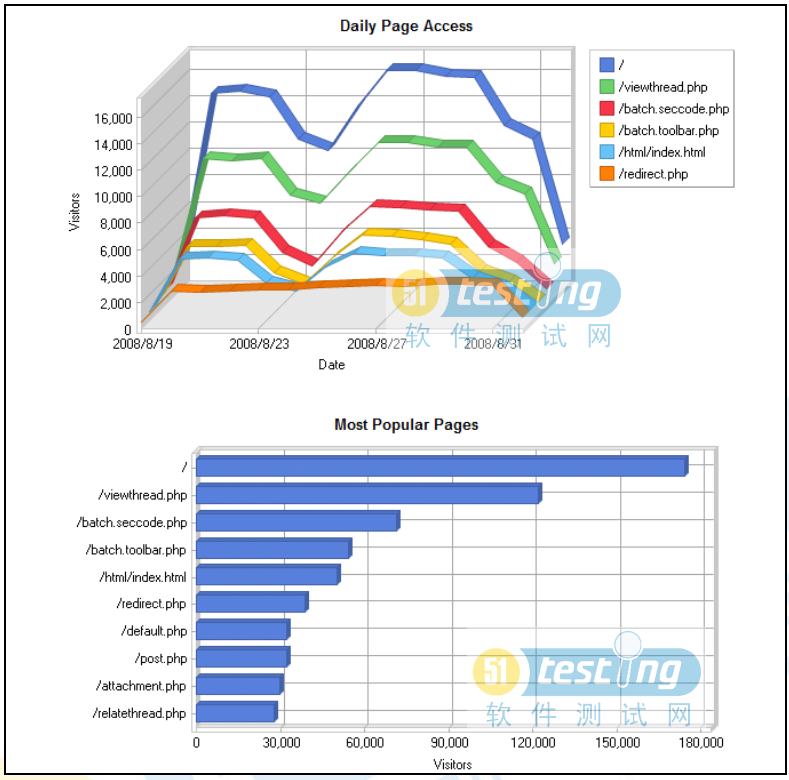


图 6.27　每日页面访问趋势和最热门访问页面

带宽较小以及网络延迟较高的用户，自然访问时间会比较长，而和服务器在同一城市的用户自然访问会比较快。在图 6.28 中可以发现系统的主要用户来自中国，如果希望进一步得到中国城市访问量的分布可以导出数据根据 IP 规划规则进行进一步分析。这里列出系统访问量最高的 20 个国家和地区记录，如表 6.5 所示。

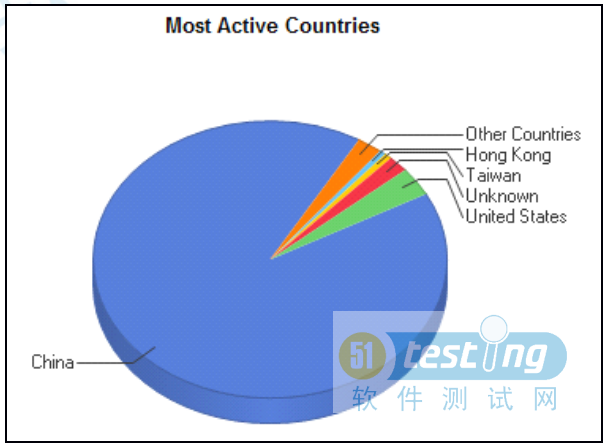


图 6.28　访问用户所属国家统计

表 6.5 客户访问国家表

Most Active Countries					
	Country	Hits	Visitors	% of Total Visitors	Bandwidth (KB)
1	China	27,336,596	530,398	91.22%	1,863,833,460
2	United States	1,724,371	19,436	3.34%	21,585,653
3	Unknown	431,567	10,907	1.88%	23,211,312
4	Taiwan	81,371	3,948	0.68%	5,452,882
5	Hong Kong	84,094	3,400	0.58%	4,551,714
6	Japan	34,102	1,556	0.27%	2,162,643
7	Singapore	32,681	1,543	0.27%	1,509,533
8	Korea, Republic of	21,898	1,426	0.25%	1,037,077
9	Canada	57,922	1,229	0.21%	2,799,533
10	Sweden	2,782	1,143	0.20%	159,666
11	Australia	21,487	921	0.16%	1,971,493
12	Germany	14,313	649	0.11%	354,962
13	France	7,339	546	0.09%	1,097,064
14	India	11,721	518	0.09%	273,660
15	United Kingdom	7,299	488	0.08%	434,769
16	Malaysia	4,386	337	0.06%	2,797,861
17	Russian Federation	13,638	218	0.04%	324,386
18	Macau	3,566	192	0.03%	614,537
19	Italy	799	154	0.03%	698,874
20	Norway	2,603	153	0.03%	24,476

对我们来说，在确定了系统的主要客户群以后，性能测试也要兼顾这些地区的网络特点进行测试，从而得到真实的响应时间，避免出现自己访问很快，但是主要客户群访问很慢的尴尬情况。

这里涉及一些前端性能分析的内容，更多内容大家可以参考第 6.1.5 节。

最后在日志中还能得到系统运行的错误统计。这些错误日志可以帮助我们分析系统存在的问题，另一方面，当进行性能测试后，也可以通过它来了解系统出错的原因。

衡量性能测试的有效性和真实性也可以通过日志来实现。如果性能测试后得到的日志分析结果和真实的历史日志数据接近，那么就可以说明性能测试几乎完全模拟了用户的行为，性能负载生成是成功有效的。所以日志是一个非常大的宝库，当做好数据挖掘后，可以从中得到大量的需求，来指导性能测试的设计与执行。

有一点请不要遗忘，就是系统负载是动态的。某些系统的客户是固定的，不会有太大的变化。例如公司内部采用 C/S 架构的系统，那么这种系统负载是可控的，而如果是一个对外访问的 B/S 架构网站，可能会存在爆发式的用户增长。在进行性能测试的时候就需要考虑这种情况，对性能测试的数据有一个容量规划，确保系统在不可控的增长下也能正常使用。而用户的增长可以参考日志的变化规律，通过一个模型来预估未来的容量。

例如：通过日志可以得到该系统从 2004 年 3 月 1 日至 2009 年 5 月 1 日的访问量统计，如表 6.6 所示。

表 6.6　客户访问量月度表

月份流量	访 问 量	月份流量	访 问 量
2004/3/1	450 (0.0%)	2006/3/1	2730991 (1.1%)
2004/4/1	160 (0.0%)	2006/4/1	2108003 (0.8%)
2004/5/1	2607 (0.0%)	2006/5/1	2048282 (0.8%)
2004/6/1	23260 (0.0%)	2006/6/1	3134630 (1.2%)
2004/7/1	275879 (0.1%)	2006/7/1	3939431 (1.5%)
2004/8/1	401205 (0.2%)	2006/8/1	3760597 (1.5%)
2004/9/1	438679 (0.2%)	2006/9/1	3287111 (1.3%)
2004/10/1	505053 (0.2%)	2006/10/1	3674063 (1.4%)
2004/11/1	853294 (0.3%)	2006/11/1	3281415 (1.3%)
2004/12/1	1692839 (0.7%)	2006/12/1	3139022 (1.2%)
2005/1/1	737470 (0.3%)	2007/1/1	2497596 (1.0%)
2005/2/1	533578 (0.2%)	2007/2/1	2703398 (1.1%)
2005/3/1	1172738 (0.5%)	2007/3/1	5396896 (2.1%)
2005/4/1	894274 (0.4%)	2007/4/1	5434648 (2.1%)
2005/5/1	896974 (0.4%)	2007/5/1	5476112 (2.1%)
2005/6/1	836325 (0.3%)	2007/6/1	7412478 (2.9%)
2005/7/1	1690384 (0.7%)	2007/7/1	8685852 (3.4%)
2005/8/1	1337772 (0.5%)	2007/8/1	8597555 (3.4%)
2005/9/1	1449475 (0.6%)	2007/9/1	8609857 (3.4%)
2005/10/1	920675 (0.4%)	2007/10/1	4982459 (2.0%)
2005/11/1	1539172 (0.6%)	2007/11/1	6960950 (2.7%)
2005/12/1	1414815 (0.6%)	2007/12/1	8799026 (3.5%)
2006/1/1	979908 (0.4%)	2008/1/1	7659880 (3.0%)
2006/2/1	1528678 (0.6%)	2008/2/1	6019874 (2.4%)
2008/3/1	7087550 (2.8%)	2008/11/1	6779644 (2.7%)
2008/4/1	7328250 (2.9%)	2008/12/1	7789938 (3.1%)
2008/5/1	7363950 (2.9%)	2009/1/1	6162039 (2.4%)
2008/6/1	6523510 (2.6%)	2009/2/1	6976834 (2.7%)
2008/7/1	7962979 (3.1%)	2009/3/1	7291965 (2.9%)
2008/8/1	7185742 (2.8%)	2009/4/1	6054961 (2.4%)
2008/9/1	8300554 (3.3%)	2009/5/1	9214862 (3.6%)
2008/10/1	8827155 (3.5%)		

转化为折线图后便可进行分析，如图 6.29 所示。

通过对历史数据的分析，2007 年 3 月至 2007 年 7 月系统的月访问量有爆发式增长，而在随后的两年内访问量比较稳定（在 1 千万以内），从容量预估的角度来说，该系统的访问量正常情况下保持在 1 千万以内，最高容量预估每月 1 千 1 百万访问量。使用同样的方法可以对在线用户数及系统数据库容量进行规划。

有关容量规划的内容，请参考相关书籍。

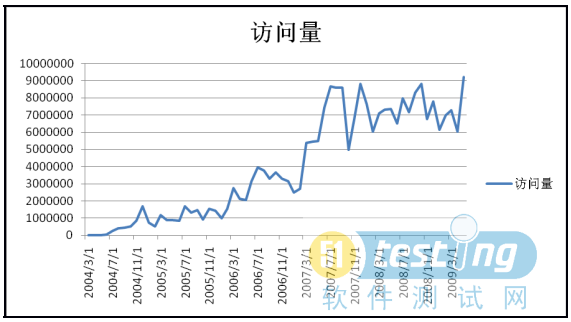


图 6.29 系统访问量发展趋势

通过参考同类业务

如果需要新建一个项目，而从前又没有相关的历史记录，前面通过日志的方法就不能使用了，不过这时可以考虑参考一下同类的业务。

虽然是第一次做这类项目，但是并不代表别人没做过，可以参考一下别人是怎么做的。例如本书一开始的奥运会订票系统，对于国内的软件开发公司来说是第一次做如此规模的 B2C 项目，但是前几届的奥运会别人也做过订票系统，问问架构和流量，如果不好意思问别人奥运会订票系统怎么做，那就问问国内火车在线订票网站，或者在线的 B2C 网站也行啊。所以考试前大家经常做的事情就是找老师打听题型和做一些以前的模拟试卷，了解一下可能会考的知识点，这也是参考同类业务的一种体现。在确认了项目的类型后，可以参考一下同类型网站的相关数据，通过这种方式来明确项目的需求。

例如：如果希望自己做一个类似 51Testing 规模的技术论坛，那么在进行性能测试时需要达到的性能指标是什么呢？答案很简单，打电话给 51Testing 说想要购买一个首页广告位，了解一下访问量和相关业务即可。

通过 80/20 原则

80%的功能只会有 20%的用户访问，反过来也可以说 80%的用户只使用 20%的功能。对于性能测试来说并不需要确保整个系统所有的功能都能完全满足性能需求，而是应该把精力集中在用户最常使用的功能上。

通过这个规则结合前面的日志或者同类项目用户的访问特征，可以对系统的功能进行划分，将不太常用的功能删除，得到最终的性能需求。

通过上面的几种方法还可以根据国家或行业规范标准来确定需求，从而获得 Discuz 论坛性能测试目标，包括目标场景和手工场景的场景模型，以及性能测试的主要业务。

明确定义概念

通过上面的性能需求分析得到了关键的用户需求规格，接着需要将需求规格转化成性能测试工具中的目标。

✓ 明确定义概念需要知道测试的最终目标是什么。

……此处内容略

搭建测试环境

在编写脚本的同时，执行场景之前需要完成测试环境的搭建工作，这里包括硬件和软件环境的搭建。根据性能测试计划中的测试环境规划，完成对整个测试环境的搭建（请协调开发人员及系统管理员等相关角色帮忙完成环境搭建的工作）。

由于性能测试的特殊性，整个测试环境需要在严格的独立监控下管理，避免不受控的情况出现导致性能测试数据的偏差（类似于药品制造中的无菌室概念），而另一方面，在实际应用中很难得到真实的系统环境来完成性能测试。很多时候不可能在研发阶段就在公司搭建出和用户上线时完全相同的测试平台，那么得出来的数据就和真实数据有一定的差距，解决这个问题一般有以下两种解决方式（预估方式均有误差）：

1．通过建模的方式实现低端硬件对高端硬件的模拟

通过配置测试来计算不同配置下的硬件性能和系统处理能力的关系，从而推导出满足系统性能的真实配置情况，这种模拟需要精确的建模，模型的采样点越多，那么得到的结果越精确，从而将在低端配置下的性能指标通过该模型转化为高端配置下的最终预估性能指标。

例如：搭建一个低端环境，首先对这个环境的 CPU 或内存进行单独的性能基准测试。例如使用 EVEREST 工具计算一下这个 CPU 的得分，再通过性能负载工具得到当前系统的 TPS 及响应时间。然后超频或者更换 CPU，再次计算相关分数，最后可以得到表 6.7。

表 6.7　硬件性能建模表

硬件配置	基准分数	TPS	响应时间
INTEL P4 1.8GB	1800	2	1.8s
INTEL P4 2GB	2000	2.2	1.8s
INTEL P4 3GB	3000	3	1.8s

通过这个表，可以简单认为每 900 基准分数约等于 1 个 TPS，而如果系统的最终需求是能够满足 20 个 TPS 的话，也就是说硬件配置的 CPU 至少需要达到 18000 分，反过来就可以知道服务器应该需要使用什么样的 CPU。

2．通过群集的方式计算

对于较大的系统来说，单台服务器的处理能力是有限的，通常都会采用群集的方式来进行负载均衡，完成对海量请求的处理。虽然无法获得整个群集的测试环境，但是可以对群集上的一个节点进行性能测试，得出该节点的处理能力，再计算每增加一个节点的性能损失，同样也可以通过建模的方式得到大型负载均衡情况下的预估性能指标。

例如：首先在单台服务器上获得具体的性能指标，每台服务器能够承受 2000 人在线、平均 TPS 为 80、响应时间为 2 秒。接着，添加负载均衡策略，再次测试负载策略下的数据损耗。得出数据后添加 1 台负载均衡服务器，测试在两台服务器下每台服务器的性能指标，以此类推，可以得到表 6.8 的数据。

表 6.8　服务器负载性能建模表

负载服务器个数	平均每台负载服务器在线用户数	平均 TPS	响应时间
1	2000	80	2

1（添加负载均衡策略）	1950	78	2
2	1950	78	2
3	1900	76	2
4	1900	75	2

随着负载均衡服务器的添加，平均每台服务器的处理能力会逐渐稳定，从而了解在什么样的情况下需要多少台负载均衡服务器。

受到环境限制，在这里搭建一个简易的测试环境，由一台较早的 PC 作为服务器而另外一台笔记本作为负载生成来模拟用户行为，完成性能测试。除了搭建硬件环境外，还需要对软件环境进行搭建，其中最重要的是环境的备份。在进行性能测试时，为了保证测试结果数据的客观公正，在每次测试前，都需要保证相同的软件环境，所以对测试环境的备份工作是非常重要的。建议使用 Ghost 对测试环境进行镜像，每次测试后还原被测环境重启系统，避免由于磁盘碎片和前一次测试的缓存影响测试结果。

对于整个测试环境的搭建，建议生成专门的文档进行管理，并进行配置管理，确保对测试环境做到基线控制。

对于性能测试环境的搭建，还有一个麻烦的问题，就是性能测试数据如何生成？也就是大家经常遇到的容量问题。在性能测试计划中存在着容量为每个版块 2 万条，共计 10 万条帖子的数据要求，如何快速有效地生成这些容量呢？

某些公司会将历史业务数据通过导入的方式实现容量的模拟，但是这种做法存在着几个问题：

1．某些数据内容敏感，是否会涉及信息泄露；

2．历史业务数据和新系统的结构不同，导出困难。

这样做不行，难道自己再编写个 LoadRunner 脚本专门来发那么多帖子么？10 万个帖子，如果用 LoadRunner 来进行数据生成效率太低，这里可以考虑使用编写 SQL 代码或者专门的数据生成工具来实现容量的生成。

通过编写一个存储过程来完成对数据的随机生成，只需要修改中间的 Insert 命令即可完成表格的适应以及对生成记录条数的控制。这里以 T-SQL 为例，该存储过程可以实现随机生成 10～20 位大小写混合的字符串，最终通过 insertrandstr 存储过程，实现记录的添加。

```
--生成随机数据
if exists (select * from sysobjects where name = 'createsinglechar' and type = 'p')
drop proc createsinglechar
go
if exists (select * from sysobjects where name = 'createsingleip' and type = 'p')
drop proc createsingleip
go
if exists (select * from sysobjects where name = 'createwholestr' and type = 'p')
drop proc createwholestr
go
if exists (select * from sysobjects where name = 'insertrandstr' and type = 'p')
drop proc insertrandstr
go
use [projcet]
go
```

```
    create procedure createsingleip
    @randip int output
    as
    declare @singleip int
    set @singleip = cast(rand()*256 as int)
    --print @singleip
    while (@singleip>255 or @singleip<0)
    begin
    set @singleip = cast(rand()*256 as int)
    --print @singleip
    end
    set @randip = @singleip
    --print @randip
    go

    create procedure createsinglechar
    @randchar varchar(3) output
    as
    declare @singlechar varchar(10)
    set @singlechar = cast(rand()*123 as int)
    while (@singlechar<65 or @singlechar>122 or (@singlechar>90 and
@singlechar<97))
    begin
    set @singlechar = cast(rand()*123 as int)
    end
    set @randchar = @singlechar
    go
    --97 122
    --65 90
    create procedure createwholestr
    @wholestr varchar(20) output
    as
    declare @len varchar(10)
    set @len = cast(rand()*20 as int)
    while (@len>20 or @len<10)
    begin
    set @len = cast(rand()*20 as int)
    end
    declare @singlechar varchar(10)
    declare @mystr varchar(20)
    declare @i int
    set @i=1
    set @mystr = ''
    while(@i<=@len)
    begin
    exec createsinglechar @singlechar output
    --select @singlechar
    set @mystr = @mystr + char(@singlechar)
    set @i=@i+1
    end
```

```
    set @wholestr = @mystr
    go
    --exec createwholestr abc

    create procedure insertrandstr
    as
    declare @data1 varchar(20)
    declare @data2 varchar(20)
    declare @ip1 int
    declare @ip2 int
    declare @ip3 int
    declare @ip4 int
    declare @ip varchar(15)

    create table testtable(
    id int not null,
    username varchar(20) not null,
    message varchar(20) not null,
    logindate varchar(20) not null,
    ip varchar(20) not null
    )
    declare @j int
    set @j=1
    while (@j<=10)
    begin
    exec createwholestr @data1 output
    exec createwholestr @data2 output
    exec createsingleip @ip1 output
    exec createsingleip @ip2 output
    exec createsingleip @ip3 output
    exec createsingleip @ip4 output

    set @ip=cast(@ip1 as varchar(3))+'.'+cast(@ip2 as varchar)+'.'+cast(@ip3 as
varchar)+'.'+cast(@ip4 as varchar)

    insert   into   testtable([id],[username],message,logindate,ip)   values
(@j,@data1,@data2,getdate(),@ip)
    set @j = @j + 1
    end
    select * from testtable
    --drop table testtable
    go

    exec insertrandstr
```

下面我们简单对存储过程做个解释。Createsingleip 存储过程生成 IP 地址格式的数字，每次调用都会返回一个 0～255 的随机数字；Createsinglechar 存储过程生成一个 a～z 或 A～Z 的随机字符；Createwholestr 存储过程调用 createsiglechar 存储过程生成 10～20 位长度的字符串；Insertrandstr 存储过程新建一张 testtable 表，然后反复调用前面三个存储过程，进行插入操作，完成数据的添加。

无论如何通过 SQL 生成数据在维护和开发上总是比较麻烦的，这里推荐一个非常好用的数据生成工具 DataFactory 来解决环境搭建中需要进行大容量生成的问题。

DataFactory 是一种快速的、易于产生测试数据的带有直觉用户接口的工具，它能建模复杂数据关系。DataFactory 是一种超强的数据产生器，它允许开发人员和 QA 很容易产生百万行有意义的正确的测试数据库，DataFactory 首先读取一个数据库方案，用户随后单击鼠标产生一个数据库。该工具支持 DB2、Oracle、Sybase、SQL Server 数据库，并且还支持 ODBC 的连接方式，如果使用的系统是 MySQL 的话，那么是无法直接使用 DataFactory 来生成数据的。

那么如何使用 DataFactory 连接 MySQL 数据库呢？虽然 DF 没有提供对 MySQL 的直接连接，但是可以通过 ODBC 的方式来过渡，这里需要注意 MySQL 默认不支持 ODBC，所以先安装 MyODBC 工具来实现 ODBC 和 MySQL 的连接，再让 DF 通过 ODBC 的方式连接 MyODBC，从而实现曲线救国。

由于计划制订的性能测试目标中需要构建 10 万条帖子记录，那么我们来看看如何使用 DataFactory 快捷地完成数据的构建工作。

打开 DataFactory，新建一个数据库连接，设置 DataFactory 连接数据库类型为 SQL Server，如图 6.30 所示。

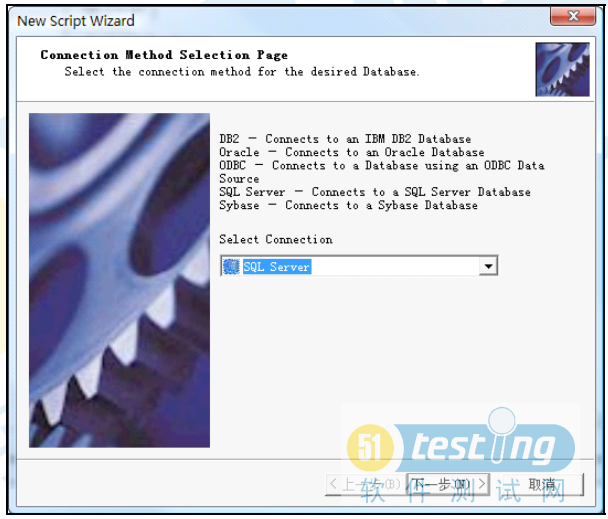


图 6.30　DF 中添加一个 SQL Server 数据库连接

在 DataFactory 中输入数据库的连接信息，如图 6.31 所示。

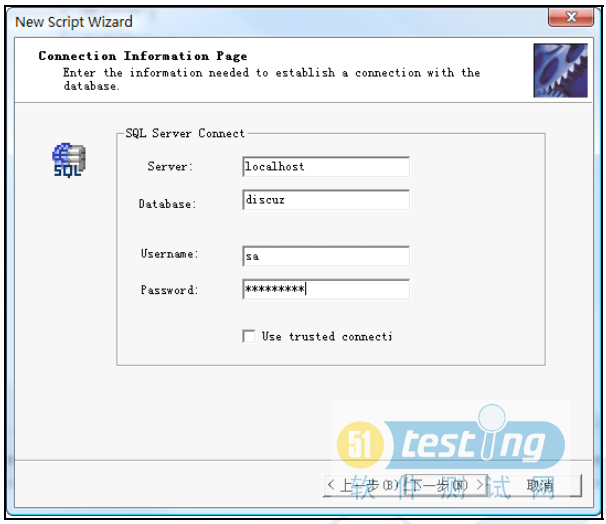


图 6.31　在 DataFactory 中设置 SQL Server 数据库连接信息

检测通过后，DataFactory 会列出该数据库中所有的表，选出需要注入数据的表后，就可以设置对于该表的数据插入数目和数据格式了。添加 dnt_posts1 和 dnt_topics 表，如图 6.32 所示。

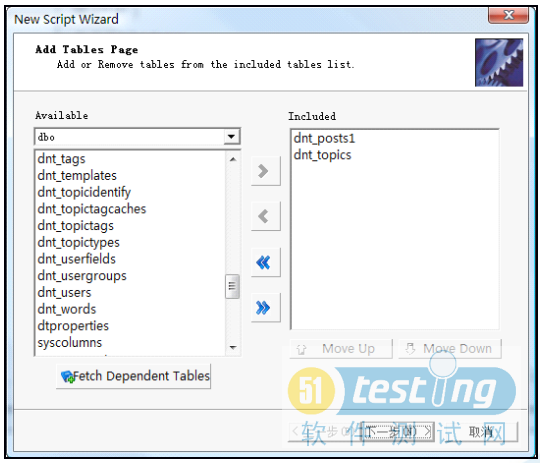


图 6.32　DataFactory 中选择需要生成数据的表名

首先根据 Discuz 论坛的数据字典了解保存帖子数据的格式，为我们通过 DataFactory 生成有效的论坛帖子记录提供支持。

根据数据字典，可以了解到当新增一个帖子时，数据库中的两张表会有直接的影响，其中 dnt_post1 表记录了帖子的正文内容，而 dnt_topics 表记录了每个板块的帖子列表，最终可以得到需要操作的属性列表，以及 DataFactory 的取值设置，如表 6.9 所示。

表 6.9　DataFactory 数据生成格式表

dnt_topics		
属性名	属性含义	数据规则
Fid	板块 id	随机 2～7
Poster	发帖人	Admin
Posterid	发帖人编号	1
Title	帖子标题	随机 40 长度字符串
Lastpostid	最后回复人编号	1
Lastposter	最后回复人	Admin
Tid	帖子标题 id	每次+1
dnt_posts1		
属性名	属性含义	数据规则
Pid	帖子正文 id	每次+1
Fid	板块 id	随机 2～7
Tid	帖子标题 id	每次+1
Parentid		1
Poster	发帖人	Admin
Posterid	发帖人 id	1
Title	帖子标题	随机 40 长度字符串
Message	帖子正文	随机 512 长度字符串
htmlon		1

接着按照每个字段的属性设置对应的取值方式。

例如 Fid 帖子所在的板块编号，论坛中有 6 个板块，编号为 2 至 7，现将这个值设置为随机取值，取值范围为“Between 2 and 7”，如图 6.33 所示。

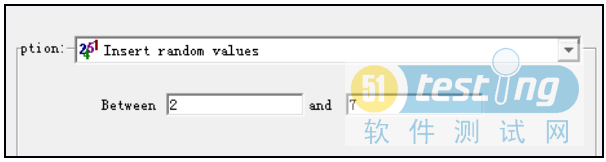


图 6.33　DataFactory 中设置数据生成为 2～7 随机数

而 pid 及 tid 设置为从 1 开始每次值加 1，要注意的是 dnt_topics.tid 在系统中设置为自动增长类型，需要强制修改该属性删除自动标识，才能确保生成的数据是有效的，如图 6.34 所示。

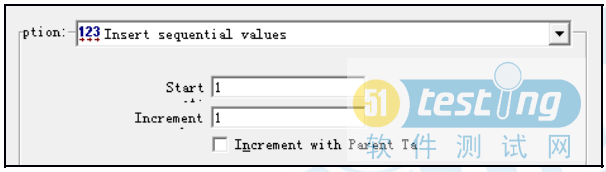


图 6.34　DataFactory 中设置数据生成为从 1 开始每次加 1

而对于发帖用户 Poster 可以设置为管理员发帖，如图 6.35 所示。

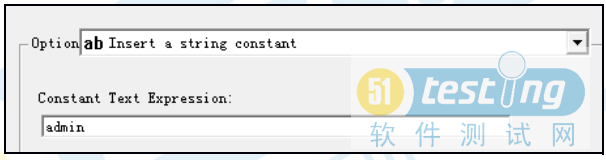


图 6.35　DataFactory 中设置数据格式为字符串 admin

将相关属性设置完成后，选择创建 100000 条记录，如图 6.36 所示。

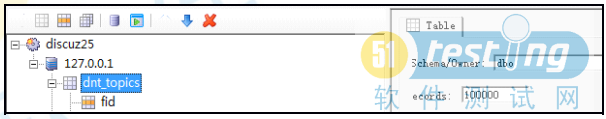


图 6.36　DataFactory 中设置对表生成的记录为 10 万条

单击 Run 按钮，稍等片刻，就在论坛中导入了 100000 条帖子记录，接着可以通过 SQL 查询一下该表的记录，如图 6.37 所示。

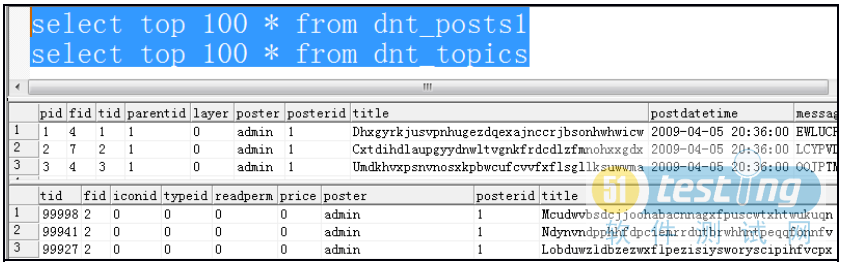


图 6.37　检查数据库中的记录已经生成

可以看到记录都已加入到表中，再去论坛上检查一下，会看到每个版块都多了很多帖

子，但是在统计上还存在一些问题，这是因为受论坛自身缓存机制的影响，在后台重置一下板块数据即可，最终效果如图 6.38 所示。

图 6.38　论坛中的数据已经生成

如果没有办法很好地得到数据字典，也可以通过 Sybase PowerDesigner 逆向工程导出数据库中的数据字典，也可以直接通过 PowerDesigner 快速生成大量已注册用户，为后面的测试脚本提供相关数据。

容量生成后记得进行及时的备份，方便测试后的还原工作。

……此处内容略

6.1.5　分析性能数据

当场景运行结束后，通过 Analysis 组件可以得到对应的性能测试报告，但是光靠这个报告并不能反映任何问题，还需要人为地对这些数据进行分析，便于进行瓶颈定位和下一步优化的参考。

性能调优原理

性能瓶颈定位和调优是一个复杂的过程，就好像改善春运状况一样，并不是简单地多修几条铁路、多开几部临客就能解决的。对于整个软件系统的性能来说，最终都能反映到单体用户的响应时间上，那么响应时间又是由哪些内容组成的呢？先思考一个问题：我每天早上坐交通工具上班要花费多少时间？这个时间的组成部分有哪些？如何调优？

大家会发现对于自己来说如果采用的交通工具单一且稳定的话，那么时间是比较固定的。例如坐地铁从 a 到 b 点的时间误差是非常小的，而选择公交大家都体会过堵车的痛苦。整个上班时间一般由以下 3 部分组成：

- 家到交通工具
- 交通工具到达目标地
- 目标地进入公司

如果上班经常迟到，那么唯一能做的就是改变交通工具，例如本来坐公交的换出租或者换地铁，其实这就是一种性能调优，通过增加成本来解决性能问题。不过你是否也遇到

过打车也堵车的尴尬呢？

我们现在先来做一个上班的性能分析及调优项目，目的就是想能够晚出门早到公司，那么首先来看看现在的情况，假设调优前 7：30 分出门，走 15 分钟到达公交车站，一般等待 5 分钟上公交（偶尔会挤不上公交），即在正常情况下会在 7：50 上公交，然后坐 50 分钟公交，在 8：40 分下公交，走 8 分钟到公司，等 5 分钟电梯，所以会在 8：53 分到达公司。这是通常情况下的状态，所以总计需要消耗大约 83 分钟在上班路上。

调优方法 1：

首先需要对所有内容进行分析，确定哪些是可以优化的，哪些不能优化，如表 6.11 所示。

表 6.11　最易调优分析表

几乎不能优化的内容	
公交车的 50 分钟	这个不是你说了算的，无法优化
等车的 5 分钟	
比较难优化的内容	
等电梯的 5 分钟	如果楼层低可以考虑自己走上去(6 楼以下的时间可以控制在 3 分钟以内)
可以优化的部分	
出门走到公交车站的 15 分钟	如果这两个时间都由走改成小跑，那么到公交车站可以节约 7 分钟，下车到公司节约 4 分钟。所以可以使用这种方式来进行一定的优化，优化效果是 11 分钟
下公交走到公司的 8 分钟	

调优方法 1 总结：

通过分析上班路程上的时间，确定可以优化的内容，进行调优，最终节约了 11 分钟。这种方法方便快捷，但是效果并不一定很好，而且经常不能解决本质问题，所以还需要考虑别的方法。

调优方法 2：

首先分析最值得调优的部分，如表 6.12 所示。

所有的时间中坐公交的时间是最长的，那么调优就从最浪费时间的部分开始。

交通工具并不只有一个，如果采用更快的交通工具那么就可以有效解决这个问题。在公司和家之间有没有更快捷稳定的交通工具呢？地铁！那么估计一下如果换乘地铁所需要花费的时间（从坐上公交到地铁站需要消耗大概 4 分钟的时间，等地铁大概需要 3 分钟，坐地铁需要 24 分钟，从地铁出站走到公司需要 8 分钟），调优结果如表 6.13 所示。

表 6.12　消耗时间分析表

出门到公交	15 分钟
等待公交	5 分钟
坐公交	50 分钟
下公交到公司	8 分钟
坐电梯	5 分钟

表 6.13　最大效益调优分析表

公交到地铁	4 分钟

等待地铁	3 分钟
坐地铁	24 分钟

我们可以发现在交通工具上所消耗的时间为 31 分钟，比乘坐公交所需要消耗的时间减少了 19 分钟，而且这个时间相对于公交稳定很多，因为公交 50 分钟是平均值，离散率是比较高的，经常会堵车，但是地铁的稳定性要高很多。

调优后成本提高了，因为要多坐一次地铁，本来上班只需要花费 2 元钱，现在成本可能增加为 4 元。

调优方法 2 总结：

这次并不是找最容易优化的地方下手，而是找最浪费时间的部分下手，这种调优方式相对来说比较困难，需要更多的技术和经验，但优化效果比较好。

调优方法 3：

通过前面两种调优方法将上班路程所需的时间进行了优化，看看优化结果，如表 6.14 所示。

表 6.14　最易调优和最大效益调优分析表

	调优前（分钟）	调优 1（分钟）	调优 2（分钟）
出门到公交	15	8	15
等待公交	5	5	5
交通设备	50	50	31（坐公交+等地铁+地铁时间）
下交通设备到公司	8	4	8
坐电梯	5	5	5
总计	83	72	64

这两种优化没有冲突的项目，如果将这两种调优方式都结合在一起，效果又会如何呢？

如表 6.15 所示，最终的时间节约了 30 分钟，这个调优结果就比较优秀了，但这并不是第 3 种调优方式，这个结果还能调优，但是会更复杂一些。

表 6.15　最易调优和最大效益调优总和分析表

	调优前（分钟）	调优后（分钟）
出门到公交	15	8
等待公交	5	5
交通设备	50	31
下交通设备到公司	8	4
坐电梯	5	5
总计	83	53

首先可以看到出门到公交车站需要 8 分钟，等待公交需要 5 分钟的时间，而坐公交到地铁需要 4 分钟，能不能在这 17 分钟上做点什么事情呢？选择骑车！相信很多住在离地铁稍微有些距离的朋友也会做这样的选择。如果选择骑车我们再来计算一下新的时间，如表 6.16 所示。

表 6.16　通过骑车带来的优化结果分析表

	调优 1+2	骑车方案
出门到公交	8	3
等待公交	5	0
公交到地铁	4	5（算上停车时间）

可以看到通过骑车这种方式将时间再次优化了 9 分钟，现在上班的时间又被简化了，如表 6.17 所示。

表 6.17　最终调优结果

	调优结果
出门到交通设备	8
等待交通设备	3
交通设备	24
交通设备到公司	4
坐电梯	5
总计	44

最终时间为 44 分钟，也就是说我们完全可以在 8 点出门并且确保在 8 点 50 分以前到达公司。

调优方法 3 总结：

通过前面的两种调优方式分析调优的效果，我们进行综合并且对每一个部件进行分析，找出之间的联系和互相影响的部分，进行进一步的分析和尝试，最后得到调优结果，如果调优失败那么继续分析。这种分析需要更多的经验和综合考虑，当实在无法达到性能目标时，可以考虑采用这种方法。

对于一个软件系统来说，从成本和难易度的角度来说一般的调优顺序如下所示。

① 软件平台设置

通过对服务平台的设置，可以更好地利用硬件资源，在不开销任何成本的情况下提升系统性能。

② 硬件平台

硬件的调优是最简单的，通过配置测试和基准测试可以很快地确定硬件更新所带来的效果。

③ 代码或 SQL 语句

通过静态分析得到负载较大的代码或 SQL 语句，将其进行修改，降低逻辑复杂度及成本开销。

④ 架构或需求

在上述方法都无法完成性能调优时，需要考虑对架构进行一定的调整，由于调整架构所带来的影响和风险非常大，万不得已不会采取该种方法。为了避免出现最终发现性能瓶颈无法修复的问题，在设计初期就应该对架构进行科学的负载和并发测试，确定架构的正确性。

现在回到正题，请问 google.cn 和 baidu.com 两个网站同时访问哪个比较快？

遇到这个问题，大家肯定想，打开浏览器自己感觉一下就行了。是不是感觉好像百度要快一点，不过又似乎很难说，因为两个网站都很快，没有秒表可能无法确定。

打开 Httpwatch，来看看 Time Chart 功能，这个功能会帮助我们分析出访问首页的所有 HTTP 请求组成，并且还有每个请求所消耗的时间，就好像上班路上多了个秒表，来计算整个时间开销一样。

访问 baidu.com，录制得到相关数据，如图 6.51 所示。可以看到，访问 Baidu 总共有 4 个页面请求，消耗的总时间是 1.099s，发送 2265 字节，接收 6927 字节，可以发现整个 Baidu 的页面非常简洁，页面小自然应该比较快。

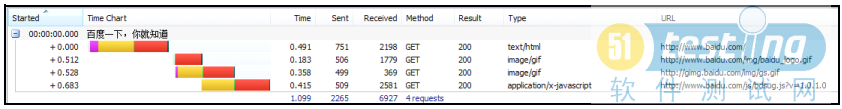


图 6.51　Baidu 页面时间分析

接着来访问一下 google.cn，如图 6.52 所示。对比百度的请求，谷歌的内容可就多了，总共包括 6 个请求，消耗总时间为 1.564s，发送 4404 字节，接收 36277 字节，相对于 Baidu 来说复杂了不少。

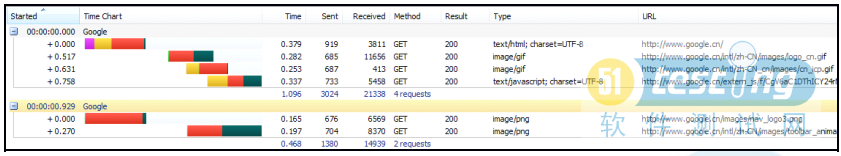


图 6.52　Google 页面时间分析

通过这样的对比，我们得到的结果是“好像”Baidu 比 Google 要快一点。不过“好像”这个词没有说服力，因为毕竟 Google 页面要大不少，慢一点也不一定能说明什么，接着进一步细化分析，对于一个网站来说，主要由以下两部分组成：

- 整个网站的 HTML 页面
- 各种图片、Java Script、CSS 和 Flash 资源

图片这类资源是不需要后台处理的，只需要 Web 服务器提供访问即可，所以先排除这类不需要处理的资源（本地缓存可以解决），针对首页的 HTML 页面进行分析。

Baidu 首页时间明细如图 6.53 所示。

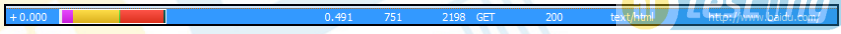


图 6.53　Baidu 首页明细

可以看到 Baidu 的首页一共花了 0.491s，其中发出请求 751 字节，收到的页面大小为 2198 字节。

接着来看看 Google 首页的时间明细，如图 6.54 所示。

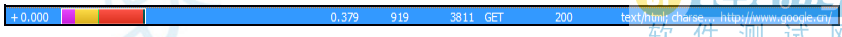


图 6.54　Google 首页明细

Google 首页只花费了 0.379s，其中发出请求 919 字节，收到页面大小为 3811 字节。

将这两份数据整理后得到表 6.18。

表 6.18　首页时间分析表

	Google	Baidu
duration	0.379s	0.491s
send	919 字节	751 字节
receive	3811 字节	2198 字节

当看到这个数据的时候，前面的分析是不是被完全推翻了？

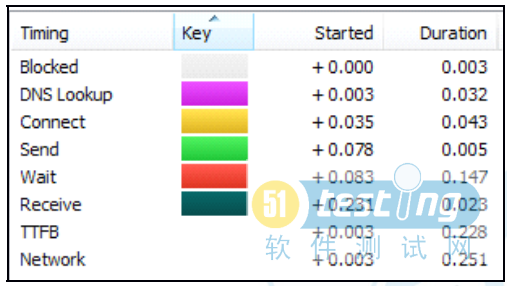
Baidu 比 Google 快？不一定了。那么为什么收发的数据较多的 Google 会比 Baidu 还快

那么一点呢？继续来分析组成首页这个静态页面的请求，到底是由哪些时间组成的。一个请求从发送给服务器到服务器返回给本地，由以下两部分组成：

- 客户端到服务器端的来回时间损耗
- 服务器端的处理时间

这个道理和大家上班一样，虽然上班时间只有 8 小时，但是在路上有人要耽误 2 个小时，有人只要 10 分钟，住在公司附近真是幸福啊。

接着来看看具体的网络时间组成，如图 6.55 所示。



Timing	Key	Started	Duration
Blocked		+0.000	0.003
DNS Lookup		+0.003	0.032
Connect		+0.035	0.043
Send		+0.078	0.005
Wait		+0.083	0.147
Receive		+0.231	0.023
TTFB		+0.003	0.228
Network		+0.003	0.251

图 6.55　网络时间组成

一个 HTTP 请求从客户端发出去经过服务器处理再返回至客户端，整个过程所需的时间（即响应时间）一般由以上这些部分组成（Blocked、DNS Lookup、Connect、Send、Wait、Receive、TTFB、Network），其中 Wait 是服务器处理业务所消耗的时间，而其他时间都是由网络损失或者别的原因导致的。通过分析我们可以发现，其实服务器的处理时间占整个响应时间的比重并不是很大，一般只占到用户响应时间的 1/6 到 1/2，这就是为什么很多朋友在局域网性能测试的时候响应时间一切正常，到了外网测试的时候就会发现响应时间慢了很多的原因。简单说，你访问国外的某个网站慢，并不一定是他们网站的业务大或者处理慢，很多时候是连接到他们服务器的网络慢。

所以性能分析和定位我们可以分成两大部分：前端性能（网络）分析和后端性能（服务）分析。

前端性能分析

首先来看看前端性能分析，除去 Wait 剩下的时间都可以当做前端分析应该处理的内容，这些组件一般消耗了整个用户响应时间的一半甚至更多，所以优化前端很多时候比优化后端更加有效且成本低廉！所谓先从影响最大的地方入手，总是最有效的。

接着首先来看看 Baidu 的前端时间组成，如图 6.56 所示。

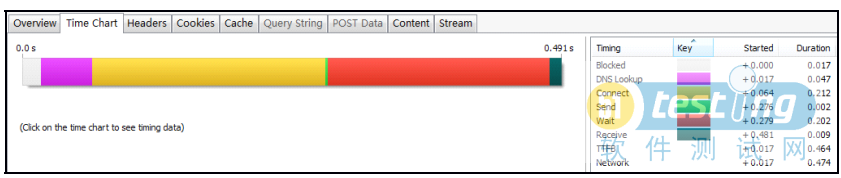


图 6.56　Baidu 首页时间细分

将数据整理后可以发现 Wait 只占了 Baidu 首页访问时间的 41%，而前端的时间开销几乎达到了 60%，如图 6.57 所示。

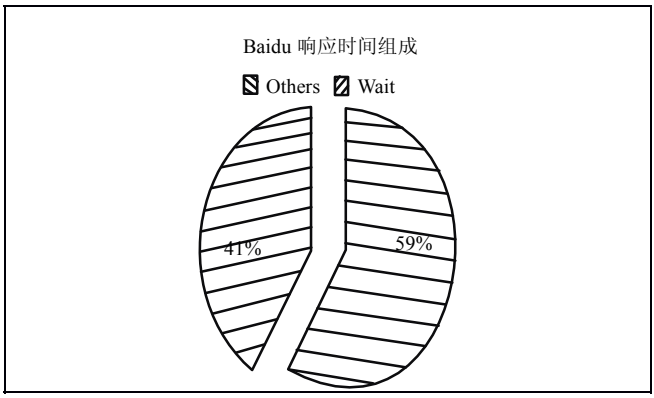


图 6.57　Baidu首页响应时间组成

而前端响应时间的内容比例如图 6.58 所示。

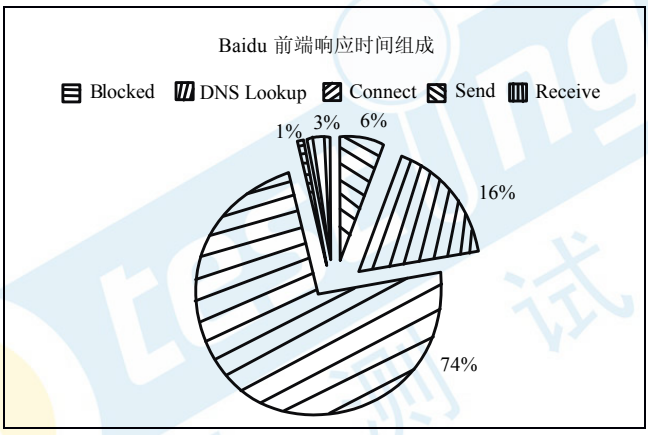


图 6.58　Baidu 前端响应时间组成

通过上面的图可以看出，Baidu 慢在 Connect 和 DNS Lookup 上，而在收发数据的时间上非常优秀，那么对比 Google 来看看怎么样呢？

Google 首页时间细分如图 6.59 所示，我们进行同样的分析操作。

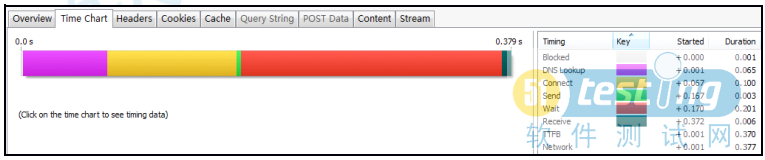


图 6.59　Google 首页时间细分

统计分析后得到图 6.60。这里对比就非常明显了，Google 页面处理的时间占总时间的 53%，而在网络上的消耗只有 47%，相比于 Baidu 来说受网络的影响较小。

再来看看前端响应时间明细，如图 6.61 所示。

可以发现时间消耗主要还是在 Connect 和 DNS Lookup 上。

作一个横向对比，可以看得更清楚，如表 6.19 所示。

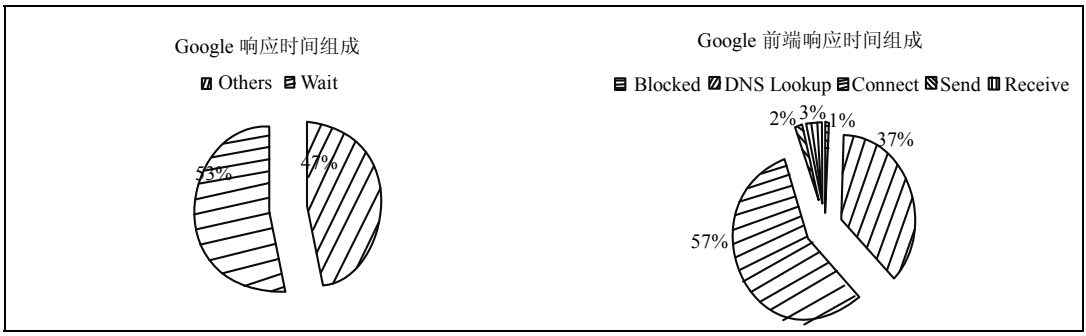


图 6.60　Google 首页响应时间组成　　图 6.61　Google 前端响应时间组成

表 6.19　首页时间组成细分表

	Google	Baidu	结果
Blocked	0.001	0.017	+0.016
DNS Lookup	0.065	0.047	-0.018
Connect	0.1	0.212	+0.112
Send	0.003	0.002	-0.001
Receive	0.006	0.009	+0.003
Others	0.175	0.287	+0.112
Wait	0.201	0.202	+0.001

转化为柱状图如图 6.62 所示。

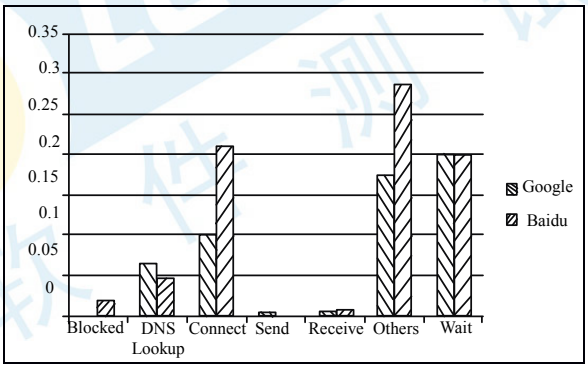


图 6.62　Google 与 Baidu 响应时间对比图

从服务器处理的速度来说 Google 和 Baidu 基本一致，从处理的内容来说，Google 略占优势。而 Baidu 前端性能最大的失败来自于 Connect，在这一项上就落后了 0.112s，所以可以确定需要优化的内容是 Connect，优化 Connect 需要更好地处理 Web 服务器的连接等待时间。而优化调优的重点是在 Connect 和 DNS lookup 上。再进一步分析，我们还可以发现更多有用的信息，如表 6.20 所示。

前面我们知道了 Google 的收发内容比 Baidu 要多，那么为什么还会出现 Receive 的时间 Baidu 反而比 Google 长呢？这个问题可能是带宽导致的，由于这里对整个时间的影响非常小，所以没有必要深究了。

表 6.20 首页网络带宽分析表

	Google	Baidu
duration	0.379	0.491
Send	919	751
Send Time	0.003	0.002
Receive	3811	2198
Receive Time	0.006	0.009

这就是前端性能分析的过程，可以发现，如果能做好前端的性能分析，并且加以优化，例如 Baidu 如果能够在 Connect 时间上达到 Google 的速度，那么两个网站的访问时间误差将从 29%回到 0%，调优的效果就十分明显了。

通过上面的介绍我们知道了如何对前端进行性能分析和定位，那么前端性能如何才能做好呢？

找到公司的网络工程师好好讨论一下吧。

现在大家知道在自己上班的过程中，除去交通工具的时间，对其他的时间怎么优化调整的了。那么平常上班的时候还会发现以下几个情况：

- 周末比平常要快
- 选择上班的时间（早出门避开高峰期）和路线
- 往前走一站虽然远但是能挤上车

前端的性能分析还有很多可以考虑的内容。例如容量规划，在不同的容量下，同样的操作会得到不同的前端性能，访问量少自然就比较快，带宽的问题不会出现。另外在路由的选择上，能否给用户提供更好的路由线路，也决定了用户的最终性能。

由于性能测试通常是在局域网内进行的，所以前端的问题并不会出现，但是通过带宽的模拟也可以得到在真实带宽下页面下载所需要的时间。当在真实环境下测试时，遇到前端问题可以使用 VisualRoute 工具帮助分析路由，也可以通过 tracert 命令来进行简单的跟踪和定位。

后端性能分析

接着来看看后端性能，在确认了前端并无明显性能瓶颈后，就要分析后端性能了。

……此处内容略