

# 用 Varnish 扩展 PHP 应用程序

通过部署反向代理扩展服务器容量

级别： 中级

Martin Streicher (martin.streicher@gmail.com), 主编, McClatchy Interactive

2008 年 4 月 03 日

使用 PHP 和反向代理（例如 Varnish）扩展 Web 服务器群的容量。

万维网 (World Wide Web) 的历史并不悠久，但是，在这个虚拟世界中充斥了许多数字垃圾。很多倒闭的 dot-com 的徽标仍在四处散播，废弃的（或者转卖的）服务器闲置着、落着灰尘，而且几乎从硅谷到硅巷的所有人都有一段精彩的经历可以讲述。“为什么，在我年轻时，我们没有优秀的所见即所得编辑器。我们手动编写 HTML，而且我们喜欢那样做！啊，那是波特时代，年轻人！”

值得庆幸的是，二十世纪九十年代中期以后，这一切有所改观。像开发人员一样，设计者可以使用优秀的工具来创建 Web 站点。使用脚本语言（包括 PHP）非常方便，而且使用诸如 CakePHP 之类的框架（请参阅 [参考资料](#)）将加快所有编码阶段的速度。站点也知道了如何扩展来跟上需求的步伐。需要更多带宽？租借一个更高带宽的通道。需要运行得更快？加快时钟周期。需要增加更多页面？部署更多 Web 服务。

需要更多服务器？也许可以做到。如果您有大量资金的话。

实际上，您可以通过许多方法扩展站点，而增加服务器只是一种（尽管通常是实用并且必要的）方法。另一项技术是重新分配现有服务器来缓和海量的传入流量。这种概念的核心是：为什么要反复地生成页面？在很多情况下，生成的页面可以存在数十秒，甚至更久。诀窍是在第 2 个、第 3 个到第 10,000 个访问者访问其 URL 时一直保持页面可用。

在这里，我将把 PHP 与名为 *反向代理* (reverse proxy) 的智能软件结合使用来缓存页面和节省服务器。

## 为什么使用反向代理？

类似计算机的内存缓存——或者类似 [PHP 操作码 \(opcode\) 缓存](#)——使用反向代理可以消除重做需求并提高常用数据的交付速度。

特别地，反向代理将在 Web 客户机与 Web 服务器之间进行仲裁，以捕捉每个传入 HTTP 请求及其各自的 HTTP 响应。然后，对于一组请求和相应的响应，反向代理可以像真正的 Web 服务器一样发挥作用。在某些情况下，反向代理可以把传入请求传递给 Web 服务器。但是在其他情况下，反向代理可以选择自己处理请求。

可以将缓存的 HTTP 响应看作一封格式化信件：反向代理仅发送一封格式化信件来响应特定请求。对某一资源（例如页面或图像）的第二个、第三个（等等）请求将收到与初始请求相同的响应（示例交换如图 1 所示）。

图 1. 虚构的反向代理与多个客户机共享相同响应

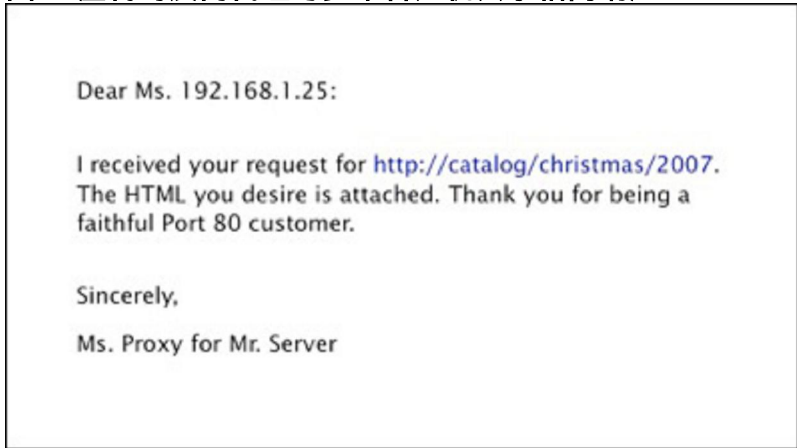
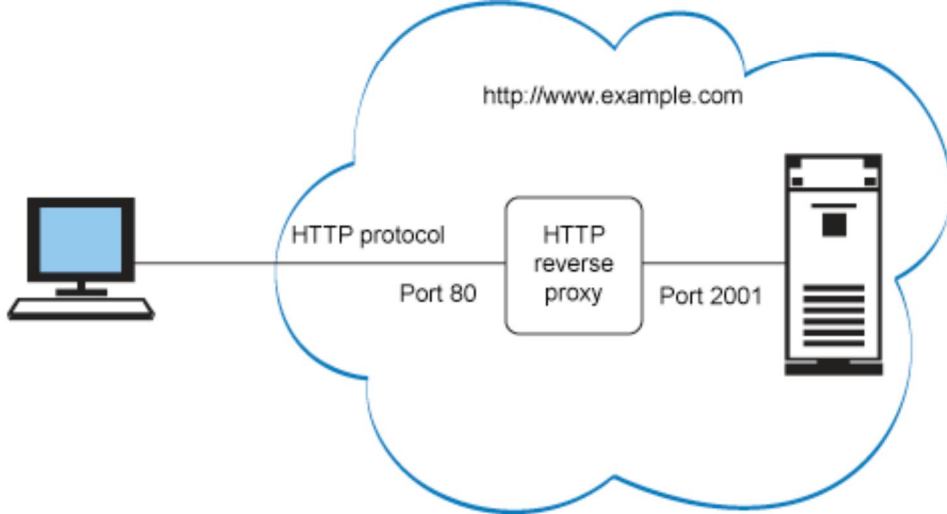


图 2 说明了客户机、服务器与反向代理之间的关系。Web 客户机——例如 Firefox 或 Safari——将连接到面向公众的 Web “服务器”的端口 80。“服务器”实际上是一个反向代理。只有代理才能通过端口 2001 连接到实际的 Web 服务器。如果代理不能完成请求或者由于缓存规则而不允许完成请求（稍后将讨论到），代理将交给 Web 服务器处理。同时，根据委托和请求类型的不同，代理可能缓存响应并将其转发给客户机。

图 2. Web 客户机、反向代理与 Web 服务器之间的关系

### 常用缩略词

- CSS**：层叠样式表 (Cascading Style Sheet)
- HTML**：超文本标记语言 (Hypertext Markup Language)
- HTTP**：超文本传输协议 (Hypertext Transfer Protocol)
- RAM**：随机存取存储器 (Random Access Memory)



除了迅速交付之外，反向代理缓存模式还提供了许多其他优点，包括：

- 反向代理维护和服务的缓存可以分担 Web 应用程序的重担。对于生成响应所需的计算，其执行次数将减少，因为期间的常见请求将由缓存来处理。
- 反向代理不会妨碍数据库服务器。大多数 Web 应用程序都依赖于数据库，因此每个请求都要求进行一次或多次查询。请求越少，转换出来的查询越少，而数据库服务器性能越好。实际上，如果数据库连接较少或者数据库服务器的响应速度较慢是整体性的，则嵌入一个或多个反向代理可以广泛获益。
- 缓存可以抵消服务器负载。较少的代码和查询将提高吞吐量。

内存是缓存的最佳持久性存储，因为访问时间（实际上）是即时的并且 RAM 的容量通常都很大（或者容易聚集在一起）。但是，文件系统也可以用作缓存存储。它比内存量大的多且成本更低，但是访问起来却非常慢。

当然，资源在 Web 上变化得非常快，并且缓存的资源最后都变得陈旧或过时。每个请求和响应都可以指定自己的“保质期”，而缓存将按计划终止每个数据，通常在捕捉后的若干钟后。

缓存的内容都是通过 *HTTP 报头* —— 每个 HTTP 请求和响应的开头来处理的。报头可以设置资源的有效期并且可以破坏缓存（客户机、服务器、代理和其他中介的复杂、微妙甚至矛盾的缓存策略超出了本文的讨论范围）。当然，HTTP 缓存控制是 HTTP V1.1 协议规范的一部分（请参阅 [参考资料](#)）。

## cache-control 报头

下面的引文提供了 HTTP 1.1 协议规范的 13.1.3 节的片段（请参阅 [参考资料](#)）。注意，对 *必须* 一词的强调并非编辑注：所有内容都是规范的一部分。

“在某些情况下，服务器或客户机可能需要为 HTTP 缓存提供显式指令。出于此目的，我们将使用 Cache-Control 报头。Cache-Control 报头允许客户机或服务器在请求或响应中传输各种指令。请求/响应链上的所有缓存机制都必须遵守 Cache-Control 指令，并且必须通过 [所有] 代理 [和] 网关应用程序。作为通用规则，如果报头值之间有任何明显冲突，则应用最严格的解释（也就是说，最有可能保留语义透明度的解释）”。

因此，如果响应包含 Cache-Control: no-cache，则反向代理或其他形式的中介 *不能* 在没有使用初始服务器重新验证响应的情况下把缓存的响应提供给其他请求。例如，如果请求包含 Cache-Control: max-age=60，则客户机声明它不愿意接受时长超过 60 秒的响应。

下面是其他一些有用的指令：

- 指令 `public` 允许把响应缓存到任何可能的位置。相反，`private` 表示响应只能存储到客户机的缓存中（通常是浏览器的缓存）。
- 指令 `must-revalidate` 强制所有缓存都验证响应。如果服务器返回 304 Not Modified 响应，表示没有发生更改，则响应仍然可用。否则，服务器将返回一个新的完整响应，该响应必须替代之前持久存储的响应。`proxy-revalidate` 指令是一个变体，它要求对公共缓存进行验证。

**注：**所有 `cache-control` 指令都可以在 HTTP 1.1 规范的 14.9 节中找到（请参阅 [参考资料](#)）。

指令也可以结合使用，如 `Cache-Control: public,max-age=30`

另外两个报头将与 Cache-Control 结合使用来控制保持力：

- `Expires` 报头用于指定格林尼治标准时间（Greenwich Mean Time, GMT）有效期。如果伴随的 Cache-Control 报头允许缓存，则 `Expires` 将控制资源可以缓存的时间。但是，如果设置了 `max-age`（或者对于公共缓存来说必然使用的 `s-`

### 元标记不足以满足需要

HTML 提供两个元标记 —— `meta http-equiv="Expires"` 和 `meta http-equiv="Pragma"` —— 来控制浏览器缓存内

maxage)，则它的值将撤销 Expires

- Last-Modified报头表示资产最后一次更改的时间 —— 同样是在 GMT 中。如前述，该报头经常用于验证缓存的内容。

因此，出于缓存 PHP 生成资源的目的，您必须设置一个或多个报头，Cache-Control Expires或 Last-Modified(要了解无法通过 HTML 完全控制缓存的原因，请参阅“元标记不足以满足需要”小节)。

容的方法。您可以把标记轻松地嵌入到任何报头中，如下所示：

```
<meta http-equiv="Expires"
  content="Wed, 26 Dec 2007 19:50:49 GMT" />
<meta http-equiv="Pragma"
  content="no-cache" />
```

但是，没有一个标记是由代理和缓存来解释的，因为没有实体解释内容。要正确设置页面，请使用合适的 HTTP V1.1 报头。

## 构建和安装 Varnish

要查看 HTTP 报头如何工作，让我们构建、安装并运行一个 HTTP 反向代理来缓存一个小型 PHP 应用程序的输出。Varnish 相对来说较新，但它是一个非常强大而且高性能的 HTTP 反向代理。要了解更多信息，请在 Varnish 维基中阅读 Varnish 的构造信息（请参阅参考资料）。Varnish 还提供了监视器和一个完整的脚本语言 Varnish Configuration Language (VCL) 来调优行为。例如，下面的代码片段将指示 Varnish 缓存通常表示静态内容的某些文件类型：

```
sub vcl_recv {
  if (req.request == "GET" && req.url ~ "\.(gif|jpg|swf|css|js)$") {
    lookup;
  }
}
```

像大多数开源包一样，Varnish 可以轻松地构建在若干个平台上，包括 FreeBSD、Linux® 和 Mac OS X。如果您希望使用包管理器（例如 apt 或 port），则也可以获得适用于一些系统的二进制形式的 Varnish。本文使用的是 Varnish V1.1.2，这是截至 2007 年 12 月的最新版本。

首先，从 Varnish Web 站点下载源代码（请参阅参考资料），解压缩 TAR 文件，然后切换到新创建的 varnish-1.1.2 目录。接下来，按顺序运行 ./autogen.sh 和 ./configure 脚本（./configure 脚本的假设通常都十分合理。但是，要自定义构建以满足系统的具体要求，键入 ./configure --help 查看可以调整哪些选项。例如，如果系统在 /opt/local 而非 /usr/local 中本地构建二进制文件，则最好运行 ./configure --prefix=/opt/local）。最后，键入 make && sudo make install。

### 清单 1. 构建和安装

```
$ ./autogen.sh
+ aclocal
+ glibtoolize --copy --force
+ autoheader
+ automake --add-missing --copy --foreign
+ autoconf

$ ./configure
checking build system type... powerpc-apple-darwin9.1.0
checking host system type... powerpc-apple-darwin9.1.0
checking target system type... powerpc-apple-darwin9.1.0
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
...
config.status: config.h is unchanged
config.status: executing depfiles commands

$ make && sudo make install
...
/usr/bin/install -c .libs/varnishd /usr/local/sbin/varnishd
...
```

在编译 C 代码的两到三分钟后，构建过程将把一些二进制代码复制到本地实用程序目录中（注意，按照约定，varnishd 将被复制到 /usr/local/sbin 中，因为它是一个系统实用程序）。

```
$ ls /usr/local/*bin/v*
/usr/local/bin/varnishadm /usr/local/bin/varnishshreply
```

```
/usr/local/bin/varnishshhist /usr/local/bin/varnishshstat
/usr/local/bin/varnishshlog /usr/local/bin/varnishshtop
/usr/local/bin/varnishshncsa /usr/local/bin/varnishshd
```

如名称所示，varnishd 文件是 Varnish 守护进程 —— 缓存和服务来自内存的内容的永久服务。上面列出的其他实用程序将控制和监视 varnishd 的操作。例如，varnishstat 持续不断地提供 Varnish 统计信息。varnishadm 文件让您在运行时向 varnishd 发送管理命令。

默认情况下，varnishd 既不缓存带有 cookie 的响应，也不遵守 Cache-Control 指令 private 和 no-cache。幸运的是，使用少量的 VCL 补丁就可以解决这个问题，如清单 1 所示。代码由 Jean-François Bustarret 提供（请参阅 [参考资料](#)）。

## 清单 2. 符合 PHP 的 VCL 片段

```
backend default {
    set backend.host = "127.0.0.1";
    set backend.port = "80";
}

sub vcl_recv {
    if (req.request != "GET" && req.request != "HEAD") {
        pipe;
    }
    if (req.http.Expect) {
        pipe;
    }
    if (req.http.Authenticate) {
        pass;
    }
    if (req.http.Cache-Control ~ "no-cache") {
        pass;
    }

    lookup;
}

sub vcl_fetch {
    if (!obj.valid) {
        error;
    }

    if (!obj.cacheable) {
        pass;
    }

    if (obj.http.Set-Cookie) {
        pass;
    }

    if (obj.http.Pragma ~ "no-cache"
        || obj.http.Cache-Control ~ "no-cache"
        || obj.http.Cache-Control ~ "private") {
        pass;
    }

    insert;
}
```

让我们继续研究代码：

- 如果不提供命令行选项 ( -b *hostname:port* )，则 backend default 部分将指定要连接的服务器。
- 当守护进程收到一个客户机请求时，将调用 vcl\_recv() 函数。反过来，当从实际的 Web 服务器检索到请求对象后或者对 Web 服务器的请求失败后，将调用 vcl\_fetch()。如前述，如果 Cache-Control 或 Pragma 报头被设为 no-cache，则 vcl\_fetch() 也拒绝缓存。
- 在代码中，pass 操作暗示“通过”，或者对于这次单独的请求/响应交换不执行任何操作。pipe 还把数据不加改变地从客户机传递到服务器，但是对客户机与服务器之间的所有后续事务都这样做（pipe 是连续不间断的 pass，直至任何一端关闭连接）。lookup 将尝试在缓存中查找响应，而 insert 将把响应添加到缓存中。

要继续，请把内容保存到文件中 —— 比如说，`/usr/local/etc/varnish/php.vcl` —— 并用以下命令启动 `varnishd`：

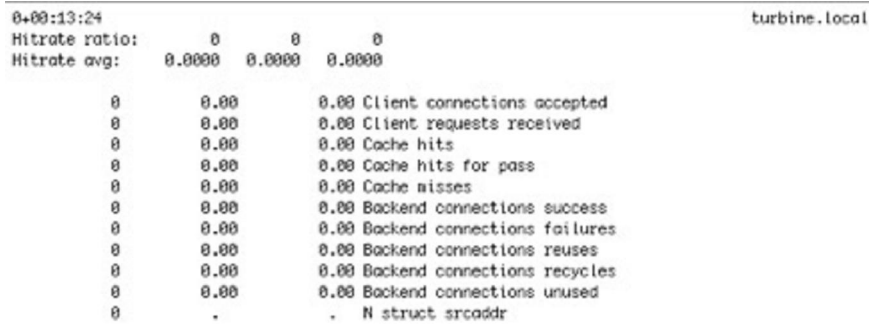
```
$ sudo varnishd -a localhost:8080 \
-f /usr/local/etc/varnish/php.vcl
```

几分钟后，您应当会看到类似下列内容的输出：

```
file ./varnish.ei npl n (unl inked)
size 1069547520 bytes (1020 fs-blocks, 261120 pages)
Creating new SHMFILE
```

`varnishd` 守护进程现在准备好进行连接。在一个终端窗口中运行 `varnishstat`。如图 3 所示，`varnishstate` 显示守护进程正在运行（运行时显示在左上角），但是还没有记录任何活动。可以在底部找到缓存中的空闲字节数。

图 3. `varnishstat` 显示缓存和连接活动



接下来，生成某个活动。连接到端口 8080 并浏览 Web 站点。进一步观看 `varnishd` 监视器。所有页面或资产是不是都显示在缓存中了？

## 使 PHP 缓存变得友好

继续之前，下载并安装最新版本的 Firefox。安装后，启动 Firefox 并访问附件页面来安装 Live HTTP Headers 插件（请参阅 参考资料）。在它的众多特性之中，Live HTTP Headers 将向您展示每个传入响应的 HTTP 报头（如果您喜欢，可以过滤对图像和 CSS 文件的请求）。根据提示重新启动 Firefox 并打开 Live HTTP Headers 窗口。

保存清单 3 中的代码，以便 Web 服务器可以找到它并使 Firefox 通过反向代理指向新 PHP 页面的地址。例如，如果 URL 是 `http://www.example.com/misc/cache.php`，则可能要指向 `http://localhost:8080/misc/cache.php`。从多个浏览器并通过 `wget` 访问同一个 URL。您应当会看到缓存正积极地发挥作用。在请求之间暂停 20 秒；您应当会看到明显的缓存丢失，因为根据它的传出报头，此内容已过期。

清单 3. 用于处理缓存的样例 PHP 代码

```
<?php
//
// Emit headers before any other content
//
cache_control( "public, max-age=10");
expires( to_gmt( time() + 10 ) );
?>
<html >
<head>
</head>
<body>
<?
    print to_gmt();
?>
</body>
</html >
<?php
function to_gmt( $now = null ) {
    return gmdate( 'D, d M Y H:i:S', ( $now == null ) ? time() : $now );
}
```



```
function last( $gmt ) {
    header("Last Modified: $gmt");
}

function expires( $gmt ) {
    header("Expires: $gmt");
}

function cache_control( $options ) {
    header("Cache-Control: $options");
}
?>
```




诚然，这个小应用程序只是比 Hello World 稍微有用一点，但是它却演示了与 Varnish 和其他反向代理协作所需的全部操作。即使动态生成大量页面，这些页面也至少可以在几秒钟内缓存下来，同时节省服务器的容量。另一方面，您可能需要阻止对其他页面进行缓存。现在您知道了规则并且拥有软件——Varnish，您可以实现一次非常有价值的优化。

## 结束语

在现代万维网中，大多数页面都不再用手动编码。相反，它们都是由应用程序生成并按需交付，实现了每个页面的自定义和个性化。

但是“天下没有免费午餐”。制作 HTML 所需的时间和工作只不过是从小转移到机器上。而且虽然机器的速度可能会快上几个数量级，但即使如此，它也是一种有限资源。聪明的 PHP 开发者意识到这种客观限制。因此计划实现扩展。数据库查询非常高效，服务器存在冗余，内存被高效使用。而现在，您可以缓存动态页面，提高通信量！

分享这篇文章.....

 [提交到 Digg](#)  
 [发布到 del.icio.us](#)  
 [Slashdot 一下！](#)

## 参考资料

### 学习

- 您可以参阅本文在 developerWorks 全球站点上的 [英文原文](#)。
- 阅读 “Make PHP apps fast, faster, fastest” 系列中的其他文章。
- 在 Varnish 维基中了解 [Varnish 的构造](#) 的更多信息。
- 详细了解 HTTP V1.1 协议规范的 [13.1.3 节](#) 和 [14.9 节](#)。
- 有关 HTTP 报头及公共中介和私有缓存行为的更多信息，请阅读 [HTTP V1.1 协议规范](#)。
- 了解 [Varnish](#) 是如何构造的。
- 有关 VCL 和 PHP 的更多信息，请访问 [Jean-François Bustarret 的博客](#)。
- [PHP.net](#) 是 PHP 开发者的重要资源。
- 查看 “[PHP 推荐读物列表](#)”。
- 浏览在 developerWorks Open source 专区中可获取的所有 [PHP 文章](#) 和 [PHP 免费教程](#)。
- 查看 IBM developerWorks 的 [PHP 项目资源](#) 以扩展 PHP 技巧。
- 收听针对软件开发人员的有趣访谈和讨论，一定要访问 [developerWorks podcast](#)。
- 要将数据库与 PHP 结合使用？查看 [Zend Core for IBM](#)，它是一个无缝的、可以立即使用、易于安装、支持 IBM DB2 V9 的 PHP 开发和生产环境。
- 随时关注 developerWorks 的 [技术事件和网络广播](#)。
- 查阅最近将在全球举办的面向 IBM 开放源码开发人员的研讨会、交易展览、网络广播和其他 [活动](#)。
- 访问 developerWorks [开放源码专区](#)，获得丰富的 how-to 信息、工具和项目更新，帮助您用开放源码技术进行开发，并与 IBM 产品结合使用。

- 查看免费的 [developerWorks On demand demo](#) 观看并了解 IBM 及开源技术和产品功能。

## 获得产品和技术

- 下载 [Varnish](#) 源代码。
- 要下载和开发 PHP 扩展，请查看 [PECL](#) 库，提供了所有已知扩展和托管设备。
- 在 [Live HTTP Headers Firefox 插件](#) 下载 Firefox 的插件。
- 使用 [IBM 试用软件](#) 改进您的下一个开发项目，这些软件可以通过下载或从 DVD 中获得。
- 下载 [IBM 产品评估版](#)，并开始使用 DB2®、Lotus®、Rational®、Tivoli® 和 WebSphere® 的应用程序开发工具和中间件产品。

## 讨论

- 参与 [developerWorks blog](#) 并加入 developerWorks 社区。
- 加入 developerWorks [PHP Forum: Developing PHP applications with IBM Information Management products \(DB2, IDS\)](#)。

## 关于作者

Martin Streicher 是 McClatchy Interactive 的首席技术官、Linux Magazine 的主编、Web 开发人员以及 developerWorks 的定期投稿者。他毕业于普渡大学并获得计算机科学硕士学位，从 1986 年起他一直从事类 UNIX 系统的编程工作。

IBM 公司保留在 developerWorks 网站上发表的内容的著作权。未经 IBM 公司或原始作者的书面明确许可，请勿转载。如果您希望转载，请通过 [提交转载请求表单](#) 联系我们的编辑团队。