

第5章

Nginx 与 JSP、ASP.NET、Perl 的安装与配置

在上一章，我们已经介绍了 Nginx 与 PHP (FastCGI) 的安装、配置与优化。Nginx 除了可以与 PHP 语言配合外，还可以和其他的一些编程语言配合。本章将介绍 Nginx 与常用的 JSP (Tomcat)、ASP.NET (Mono+FastCGI)、Perl (FastCGI) 的安装与配置。

5.1 Nginx 与 JSP (Tomcat) 在 Linux 上的安装、配置

JSP (Java Server Pages) 是由 Sun Microsystems 公司倡导、许多公司一起参与建立的一种动态网页技术标准。JSP 技术有点类似 ASP 技术，它是在传统的网页 HTML 文件 (*.htm,*.html) 中插入 Java 程序段 (Scriptlet) 和 JSP 标记 (tag)，从而形成 JSP 文件 (*.jsp)。使用 JSP 开发的 Web 应用是跨平台的，既能在 Linux 下运行，也能在其他操作系统上运行。

Tomcat 是 Apache 软件基金会 (Apache Software Foundation) 的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 5 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因此深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，它已成为目前比较流行的 Web 应用服务器。目前的最新版本是 6.0。

JDK (Java Development Kit) 是 Sun Microsystems 针对 Java 开发员的产品。自 Java 推出以来, JDK 已经成为使用最广泛的 Java SDK。JDK 是整个 Java 的核心, 包括 Java 运行环境、Java 工具和 Java 基础的类库。JDK 是学好 Java 的第一步。从 SUN 的 JDK 5.0 开始, 提供了泛型等非常实用的功能, 其版本也不断更新, 运行效率得到了很大的提高。

5.1.1 Tomcat 和 JDK 的安装

在 Linux 上, 我们首先要安装 JDK。JDK 1.6 可以在以下网址下载:

<http://java.sun.com/javase/downloads/widget/jdk6.jsp>

下载完成后, 修改 jdk-6u17-linux-x64.bin 的文件属性为可执行, 然后执行该程序安装 JDK:

```
chmod +x jdk-6u17-linux-x64.bin
./jdk-6u17-linux-x64.bin
```

按空格键看完协议, 当出现提示 “Do you agree to the above license terms? [yes or no]” 时, 输入 “yes”。安装完成后, 执行以下语句:

```
mv jdk1.6.0_17 /usr/local/jdk
vi /etc/profile
```

在文件末尾增加以下内容:

```
JAVA_HOME="/usr/local/jdk"
CLASS_PATH="$JAVA_HOME/lib:$JAVA_HOME/jre/lib"
PATH=".:$PATH:$JAVA_HOME/bin"
CATALINA_HOME="/usr/local/tomcat"
export JAVA_HOME CATALINA_HOME
```

保存并退出 vi 后, 执行以下命令使配置生效:

```
source /etc/profile
```

安装完成 JDK 之后, 按照以下步骤安装 Tomcat 二进制版本:

```
wget http://apache.freelamp.com/tomcat/tomcat-6/v6.0.20/bin/apache-tomcat-6.0.20.tar.gz
tar zxvf apache-tomcat-6.0.20.tar.gz
mv apache-tomcat-6.0.20 /usr/local/tomcat
cp -rf /usr/local/tomcat/webapps/* /data0/htdocs/www/
vi /usr/local/tomcat/conf/server.xml
```

查找 appBase="webapps", 修改为 appBase="/data0/htdocs/www", 其中/data0/htdocs/www 即为您的网页根目录。

安装完成后, 启动 Tomcat, 默认监听的是 8080 端口:

```
/usr/local/tomcat/bin/startup.sh
```

停止 Tomcat 可以使用以下命令:

```
/usr/local/tomcat/bin/shutdown.sh
```

5.1.2 Nginx 与 Tomcat 的配置

nginx.conf 配置文件内容如代码 5-1 所示。在配置文件中，静态 HTML 网页、图片、JS、CSS、Flash 等使用 Nginx 来处理，以便得到更快的速度，文件扩展名为.jsp、.do 的请求，由 Nginx 反向代理 Tomcat HTTP 服务器来处理：

代码 5-1

```
user www www;
worker_processes 8;
error_log /usr/local/webserver/nginx/logs/nginx_error.log crit;
pid /usr/local/webserver/nginx/nginx.pid;
#Specifies the value for maximum file descriptors that can be opened by this process.
worker_rlimit_nofile 65535;
events
{
    use epoll;
    worker_connections 65535;
}
http
{
    include mime.types;
    default_type application/octet-stream;
    charset utf-8;
    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 300m;
    sendfile on;
    tcp_nopush on;
    keepalive_timeout 60;
    tcp_nodelay on;
    client_body_buffer_size 512k;
    proxy_connect_timeout 5;
    proxy_read_timeout 60;
    proxy_send_timeout 5;
    proxy_buffer_size 16k;
    proxy_buffers 4 64k;
    proxy_busy_buffers_size 128k;
    proxy_temp_file_write_size 128k;
    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 2;
    gzip_types text/plain application/x-javascript text/css application/xml;
    gzip_vary on;
    upstream tomcat_server {
```

```

server 127.0.0.1:8080;
}
server
{
    listen      80;
    server_name www.yourdomain.com;
    index index.html index.htm index.jsp default.jsp index.do default.do;
    root /data0/htdocs/www;
    if (-d $request_filename)
    {
        rewrite ^/(.*)(([/])?)$ http://$host/$1$2/ permanent;
    }
    location ~ \.(jsp|jspx|do)?$ {
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_pass http://tomcat_server;
    }
    location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
    {
        expires      30d;
    }
    location ~ .*\. (js|css)?$
    {
        expires      1h;
    }
    access_log off;
}
}

```

启动 Nginx:

```
/usr/local/webserver/nginx/sbin/nginx
```

如果 Nginx 处于运行状态, 也可以使用 `nginx -t` 检查 `nginx.conf` 配置文件无错误后, 使用“kill -HUP Nginx 主进程号”来平滑重启 Nginx。

Nginx 启动后, 可以访问以下 URL 中的 JSP 示例程序, 检查 JSP 程序能否正常运行:

```
http://www.yourdomain.com/examples/jsp/
```

5.2 Nginx 与 ASP.NET (Mono+FastCGI) 在 Linux 上的安装、配置

Mono 是一个由 Novell 公司 (先前是 Ximian) 主持的项目。该项目的目标是创建一系列符合标准 ECMA (Ecma-334 和 Ecma-335) 的 .NET 工具, 包括 C# 编译器和共同语言 (CL 即 Common Language) 执行平台 (Platform)。与微软的 .NET 不同, Mono 项目不仅可以运行于 Windows 系统内, 还可以运行于 Linux、FreeBSD、Unix、Mac OS X 和 Solaris 操作系统内, 不过, 部分 Windows 上的 ASP.NET 程序迁移到 Linux+Mono 平台时须要做一些移植、修改。

实战 Nginx: 取代 Apache 的高性能 Web 服务器 www.TopSage.com

Mono 项目的官方网站为 <http://www.mono-project.com/>, 您可以通过该网站了解 Mono 的更多信息。



5.2.1 Mono 的安装

在 CentOS 5 Linux (注: 目前只兼容 32 位系统) 上, 按照代码 5-2 中的步骤安装 mono:

代码 5-2

```
yum groupinstall "Development Tools"
yum install httpd build-essential gcc bzip bison pkgconfig glib-devel glib2-devel
httpd-devel libpng-devel libX11-devel freetype fontconfig pango-devel ruby ruby-rdoc
gtkhtml38-devel wget

wget http://ftp.novell.com/pub/mono/sources/mono/mono-2.6.1.tar.bz2
tar jxvf mono-2.6.1.tar.bz2
cd mono-2.6.1/
./configure --prefix=/usr
make
make install
cd ../
```

从 SVN 版本库安装 fastcgi-mono-server, 如代码 5-3 所示:

代码 5-3

```
export PKG_CONFIG_PATH=/usr/lib/pkgconfig/:/usr/lib/
yum install subversion
svn co http://mono-soc-2007.googlecode.com/svn/trunk/brian/FastCgi/
fastcgi-mono-server
cd fastcgi-mono-server/
./autogen.sh
make
make install
cd ../
```

以 FastCGI 方式启动 fastcgi-mono-server2, 监听本机的 9001 端口, 网页根目录为/data0/htdocs/www/:

```
nohup /bin/sh /usr/local/bin/fastcgi-mono-server2 /socket=tcp:9001
/root=/data0/htdocs/www/ 2>&1 > /dev/null &
```

5.2.2 Nginx 与 ASP.NET (Mono+FastCGI) 的配置

nginx.conf 配置文件内容如代码 5-4 所示。在配置文件中, 静态 HTML 网页、图片、JS、CSS、Flash 等使用 Nginx 来处理, 以便得到更快的速度, 文件扩展名为.aspx、.asmx、.ashx、.asax、.ascx、.soap、.rem、.axd、.cs、.config、.dll 的请求, 由 Nginx 交给 fastcgi-mono-server2 进程去处理:

代码 5-4

```
user www www;

worker_processes 8;

error_log /usr/local/webserver/nginx/logs/nginx_error.log crit;

pid /usr/local/webserver/nginx/nginx.pid;

#Specifies the value for maximum file descriptors that can be opened by this process.
worker_rlimit_nofile 65535;

events
{
    use epoll;
    worker_connections 65535;
}

http
{
    include mime.types;
    default_type application/octet-stream;

    charset utf-8;

    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 8m;

    sendfile on;
    tcp_nopush on;

    keepalive_timeout 60;

    tcp_nodelay on;

    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;

    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 2;
```

```

gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;

server
{
    listen      80;
    server_name www.yourdomain.com;
    index index.html index.htm index.aspx default.aspx;
    root /data0/htdocs/www;

    location ~ \.(aspx|asmx|ashx|asax|ascx|soap|rem|axd|cs|config|dll)?$ {
        fastcgi_pass 127.0.0.1:9001;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
    {
        expires      30d;
    }

    location ~ .*\. (js|css)?$
    {
        expires      1h;
    }

    access_log off;
}

```

启动 Nginx:

```
/usr/local/webserver/nginx/sbin/nginx
```

如果 Nginx 处于运行状态,也可以使用 `nginx -t` 检查 `nginx.conf` 配置文件无错误后,使用“kill -HUP Nginx 主进程号”来平滑重启 Nginx。

Nginx 启动后,我们可以在 `/data0/htdocs/www/` 目录下下载一个名为 `info.aspx` 的 ASP.NET 探针文件,以检查 ASP.NET 程序能否正常运行:

```

cd /data0/htdocs/www/
wget http://aspnet.sysinfo.googlecode.com/files/aspnet.sysinfo-revision_23.zip
unzip aspnet.sysinfo-revision_23.zip

```

通过浏览器访问 `http://www.yourdomain.com/info.aspx`, 如果一切正常,则显示的内容如表 5-1 所示:

表 5-1 访问 ASP.NET 探针显示的部分内容

System Information

Name	Value
Server Name	localhost.localdomain
Server IP	192.168.1.2
Server Domain	www.yourdomain.com
Server Port	80
Web Server Version	nginx/0.7.27
Virtual Request Path	/info.aspx
Physical Request Path	/data0/htdocs/www/info.aspx
Virtual Application Root Path	/
Physical Application Root Path	/data0/htdocs/www/
Operating System	Linux version 2.6.18-92.el5PAE (mockbuild@builder16.centos.org) (gcc version 4.1.2 20071124 (Red Hat 4.1.2-42)) #1 SMP Tue Jun 10 19:22:41 EDT 2008 -- Unix 2.6.18.92
Operating System Installation Directory	
.Net Version	2.0.50727.1433
.Net Language	Chinese (China)
Server Current Time	10-1-10 上 12 时 56 分 18 秒
System Uptime	-16.04:25:09.1510000
Script Timeout	00:01:50

Processor Information

Name	Value
Processor 0	Intel(R) Xeon(TM) CPU 2.80GHz - 2793.466 MHz
Processor 1	Intel(R) Xeon(TM) CPU 2.80GHz - 2793.466 MHz
Processor 2	Intel(R) Xeon(TM) CPU 2.80GHz - 2793.466 MHz
Processor 3	Intel(R) Xeon(TM) CPU 2.80GHz - 2793.466 MHz

Memory Information

Name	Value
Current Working Set	0 Bytes
Physical Memory Size	4152208 kB
Physical Free Memory Size	151736 kB
Swap Total Size	8385848 kB
Swap Free Size	7679308 kB

5.3 Nginx 与 Perl (FastCGI) 在 Linux 上的安装、配置

Perl 是一种脚本语言，最初的设计者为拉里·沃尔 (Larry Wall)，Perl 语言于 1987 年 12 月 18 日发表。Perl 吸取了 C、sed、awk、shell scripting 及很多其他程序语言的特性。与脚本语言一样，Perl 不需要编译器和链接器来运行代码，要做的只是写出程序并告诉 Perl 来运行而已。这意味着 Perl 对于小的编程问题的快速解决方案和为大型事件创建原型来测试潜在的解决方案是十分理想的。Perl 的解释程序是开放源码的免费软件，使用 Perl 不必担心费用。Perl 能在绝大多数操作系统运行，可以方便地向不同操作系统迁移。

Perl 可以采用 FastCGI 方式与 Nginx 配合使用。

5.3.1 Perl (FastCGI) 的安装

在 CentOS 5 Linux 上，我们按照以下步骤安装 Perl (FastCGI)：

```
yum install perl*
perl -MCPAN -e 'install FCGI'
perl -MCPAN -e 'install FCGI::ProcManager'
vi /usr/local/bin/cgiwrap-fcgi.pl
```

然后输入代码 5-5 所示的内容：

代码 5-5

```
#!/usr/bin/perl
use FCGI;
use Socket;
use FCGI::ProcManager;
sub shutdown { FCGI::CloseSocket($socket); exit; }
sub restart { FCGI::CloseSocket($socket); &main; }
use sigtrap 'handler', \&shutdown, 'normal-signals';
use sigtrap 'handler', \&restart, 'HUP';
require 'syscall.ph';
use POSIX qw(setsid);

END() { }
BEGIN() { }
{
    no warnings;
    *CORE::GLOBAL::exit = sub { die "fakeexit\nrc=" . shift() . "\n"; };
};

eval q{exit};
if ($?) {
    exit unless $? =~ /^fakeexit/;
}
```

```

&main;

sub daemonize() {
    chdir '/' or die "Can't chdir to /: $!";
    defined( my $pid = fork ) or die "Can't fork: $!";
    exit if $pid;
    setsid() or die "Can't start a new session: $!";
    umask 0;
}

sub main {
    $proc_manager = FCGI::ProcManager->new( {n_processes => 32} ); #FastCGI 进程数
    $socket = FCGI::OpenSocket("127.0.0.1:9002", 10 ); #use IP sockets
    $socket = FCGI::OpenSocket( "/opt/nginx/fcgi/cgi.sock", 10 )
    ; #use UNIX sockets - user running this script must have w access to the 'nginx'
      folder!!
    $request =
    FCGI::Request( \*STDIN, \*STDOUT, \*STDERR, \%req_params, $socket,
    &FCGI::FAIL_ACCEPT_ON_INTR );
    $proc_manager->pm_manage();
    if ($request) { request_loop() }
    FCGI::CloseSocket($socket);
}

sub request_loop {
    while ( $request->Accept() >= 0 ) {
        $proc_manager->pm_pre_dispatch();

        #processing any STDIN input from WebServer (for CGI-POST actions)
        $stdin_passthrough = '';
        { no warnings; $req_len = 0 + $req_params{'CONTENT_LENGTH'}; };
        if ( ( $req_params{'REQUEST_METHOD'} eq 'POST' ) && ( $req_len != 0 ) ) {
            my $bytes_read = 0;
            while ( $bytes_read < $req_len ) {
                my $data = '';
                my $bytes = read( STDIN, $data, ( $req_len - $bytes_read ) );
                last if ( $bytes == 0 || !defined($bytes) );
                $stdin_passthrough .= $data;
                $bytes_read += $bytes;
            }
        }

        #running the cgi app
        if (
            ( -x $req_params{SCRIPT_FILENAME} ) &&      #can I execute this?
            ( -s $req_params{SCRIPT_FILENAME} ) &&      #Is this file empty?
            ( -r $req_params{SCRIPT_FILENAME} )          #can I read this file?
        ) {
            pipe( CHILD_RD, PARENT_WR );
            pipe( PARENT_ERR, CHILD_ERR );
            my $pid = open( CHILD_O, "-| " );
            unless ( defined($pid) ) {
                print("Content-type: text/plain\r\n\r\n");
            }
        }
    }
}

```

```

print "Error: CGI app returned no output - Executing
$req_params{SCRIPT_FILENAME} failed !\n";
next;
}
$oldfh = select(PARENT_ERR);
$| = 1;
select(CHILD_O);
$| = 1;
select($oldfh);
if ( $pid > 0 ) {
    close(CHILD_RD);
    close(CHILD_ERR);
    print PARENT_WR $stdin_passthrough;
    close(PARENT_WR);
    $rin = $rout = $ein = $eout = '';
    vec( $rin, fileno(CHILD_O), 1 ) = 1;
    vec( $rin, fileno(PARENT_ERR), 1 ) = 1;
    $ein = $rin;
    $nfound = 0;

    while ( $nfound = select( $rout = $rin, undef, $ein = $eout, 10 ) ) {
        die "$!" unless $nfound != -1;
        $r1 = vec( $rout, fileno(PARENT_ERR), 1 ) == 1;
        $r2 = vec( $rout, fileno(CHILD_O), 1 ) == 1;
        $e1 = vec( $eout, fileno(PARENT_ERR), 1 ) == 1;
        $e2 = vec( $eout, fileno(CHILD_O), 1 ) == 1;

        if ($r1) {
            while ( $bytes = read( PARENT_ERR, $errbytes, 4096 ) ) {
                print STDERR $errbytes;
            }
            if ($!) {
                $err = $!;
                die $!;
                vec( $rin, fileno(PARENT_ERR), 1 ) = 0
                unless ( $err == EINTR or $err == EAGAIN );
            }
        }
        if ($r2) {
            while ( $bytes = read( CHILD_O, $s, 4096 ) ) {
                print $s;
            }
            if ( !defined($bytes) ) {
                $err = $!;
                die $!;
                vec( $rin, fileno(CHILD_O), 1 ) = 0
                unless ( $err == EINTR or $err == EAGAIN );
            }
        }
        last if ( $e1 || $e2 );
    }
    close CHILD_RD;
    close PARENT_ERR;

```

```

        waitpid( $pid, 0 );
    } else {
        foreach $key ( keys %req_params ) {
            $ENV{$key} = $req_params{$key};
        }

        # cd to the script's local directory
        if ( $req_params{SCRIPT_FILENAME} =~ /^(.*)\[^\w/\] +$/ ) {
            chdir $1;
        }
        close(PARENT_WR);
        #close(PARENT_ERR);
        close(STDIN);
        close(STDERR);

        #fcntl(CHILD_RD, F_DUPFD, 0);
        syscall( &SYS_dup2, fileno(CHILD_RD), 0 );
        syscall( &SYS_dup2, fileno(CHILD_ERR), 2 );

        #open(STDIN, "<&CHILD_RD");
        exec( $req_params{SCRIPT_FILENAME} );
        die("exec failed");
    }
} else {
    print("Content-type: text/plain\r\n\r\n");
    print "Error: No such CGI app - $req_params{SCRIPT_FILENAME} may not exist or
        is not executable by this process.\n";
}
}
}
}

```

赋予脚本可执行权限并启动 32 个 Perl-FastCGI 进程。FastCGI 进程启动后，将监听本机的 9002 端口。

```

chmod +x /usr/local/bin/cgiwrap-fcgi.pl
cgiwrap-fcgi.pl 2>&1 > /dev/null &

```

5.3.2 Nginx 与 Perl (FastCGI) 的配置

nginx.conf 配置文件内容如代码 5-6 所示。在配置文件中，静态 HTML 网页、图片、JS、CSS、Flash 等使用 Nginx 来处理，以便得到更快的速度，文件扩展名为 .perl、.pl、.cgi 的请求，由 Nginx 交给 Perl (FastCGI) 进程去处理：

代码 5-6

```

user www www;

worker_processes 8;

```

```
error_log /usr/local/webserver/nginx/logs/nginx_error.log crit;

pid /usr/local/webserver/nginx/nginx.pid;

#Specifies the value for maximum file descriptors that can be opened by this process.
worker_rlimit_nofile 65535;

events
{
    use epoll;
    worker_connections 65535;
}

http
{
    include mime.types;
    default_type application/octet-stream;

    charset utf-8;

    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 8m;

    sendfile on;
    tcp_nopush on;

    keepalive_timeout 60;

    tcp_nodelay on;

    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;

    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.1;
    gzip_comp_level 2;
    gzip_types text/plain application/x-javascript text/css application/xml;
    gzip_vary on;

    server
    {
        listen 80;
        server_name www.yourdomain.com;
```

```
index index.html index.htm index.pl index.perl index.cgi;
root /data0/htdocs/www;

location ~ \.(pl|perl|cgi)?$ {
    fastcgi_pass 127.0.0.1:9002;
    fastcgi_index index.cgi;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}

location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
{
    expires 30d;
}

location ~ .*\. (js|css)?$
{
    expires 1h;
}

access_log off;
}
}
```

启动 Nginx:

```
/usr/local/webserver/nginx/sbin/nginx
```

如果 Nginx 处于运行状态,也可以使用 `nginx -t` 检查 `nginx.conf` 配置文件无错误后,使用“kill -HUP Nginx 主进程号”来平滑重启 Nginx。

Nginx 启动后,可以在 `/data0/htdocs/www/` 目录下创建一个名为 `test.cgi` 的 Perl 测试文件,来检查 Perl 程序能否正常运行:

```
cd /data0/htdocs/www/
vi test.cgi
```

输入以下内容:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "<html><body>Hello, world.</body></html>";
```

然后赋予 `test.cgi` 文件可执行权限:

```
chmod -R 777 test.cgi
```

通过浏览器访问 `http://www.yourdomain.com/test.cgi`, 如果一切正常,显示的内容如下:
Hello, world.