make clean

./configure ****

make

make install

centos5 的系统， 不想重新编译安装apache

怎么添加cache模块？？

一。

httpd -l

列出编译的模块文件。一般可以在/usr/local/apache2/include/文件夹下找到这里列出的文件。

二。

httpd -V

这里可以看出大部分关于apache的版本信息，安装操作系统位数平台及apr版本。

三。

httpd -M

列出编译过的模块。同时能看到哪些模块是static静态编译，哪一些是shared动态加载的。

四。

config.nice

这个文件一般在prefix所指目录例/usr/local/apache2/build/config.nice文件。用vi打开看，复制出来可以直贴到shell窗口进行configure。

如果已经static编译过的模块，再次修改httpd.conf方式用loadmodule命令，在启动apache时会报

"模块名" is built-in and can't be loaded.

的错误。

/usr/sbin/apachectl -l

Compiled in modules:

core.c

prefork.c

http_core.c

mod_so.c

以前一直有这样的需求，是说apache缺少我需要的模块，可是如何在不影响现有程序的基础上，直接添加一个loadModule呢，但是模块文件从哪里来？从另外一个系统复制一份过来是否可行？虽然只是一个 foo.so 文件，放到module文件夹就可以了，但我还是查了一下资料，找到合适的方法，并且尝试过了。

apache 添加 **mod_rewrite** 模块

我使用的是httpd 2.0.55 注意配置httpd.conf 时 是 rewrite_module 而不是 mod_rewrite

1、Apache安装**rewrite**模块的时候需要**DBM**支持，否则无法编译。使用**rpm -qa gdbm** 查询，如果没有，需要先下载并安装**gdbm**：

下载地址：**ftp://ftp.gnu.org/gnu/gdbm/**

安装步骤:

./configure

make

make install

make install-compat (如果不执行此步,无法编译出ndbm.h头文件)

2、安装

#cd /root/src/httpd-2.0.55/modules/mappers

#/usr/local/apache/bin/apxs -c mod_rewrite.c

#gcc -shared -o mod_rewrite.so mod_rewrite.o -lgdbm

#/usr/local/apache/bin/apxs -i -A -n mod_rewrite mod_rewrite.so

/usr/local/apache/bin/ 是我正在使用的apahce的目录

3、配置httpd.conf
加入 LoadModule rewrite_module modules/mod_rewrite.so

注意是 rewrite_module 而不是 mod_rewrite

过程其实很容易理解:
1 在httpd的源文件里面找到 foo.c
2 运行程序编译出 foo.so
3 在httpd.conf添加进来

这里面要注意的两点:
1 foo.c是按功能放置在modules目录的，自己找到相应目录
2 loadModule命令默认是注释的，要去掉后还要改一下名称

下面是尝试**expires**功能的添加:

cd modules/metadata/
（expires属于metadata功能部分的）
/Data/apps/apache/bin/apxs -c mod_expires.c
gcc -shared -o mod_expires.so mod_expires.o -lgdbm
/Data/apps/apache/bin/apxs -i -A -n mod_expires mod_expires.so

编辑httpd.conf文件
LoadModule expires_module modules/mod_expires.so

然后测试一下:
引用

[root@localhost apache]# ./bin/apachectl -t
Syntax OK
[root@localhost apache]# ./bin/apachectl graceful

这个模块很实用，可以给你带来带宽的节省
比如如下配置:

```
<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresDefault "access plus 6 hours"
    ExpiresByType text/html "access plus 0 hours"
</IfModule>
```

启用expires，默认缓存6小时，但不缓存 text/html
也就是访问阿权的网页

http://www.aslibra.com/blog/read.php?132 不会缓存网页
http://www.aslibra.com/blog/up/1158240093.jpg 缓存图片和脚本等

意思就是这样了

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

文本页面(htm/css/js等)启用压缩后，一般可以压缩70%左右。即50K的文件，实际只需传输15K到客户端，由客户端解压显示。另外，实践证明，启用Gzip压缩后，不会对搜索引擎收录有影响。在Apache1.3时代，有一个mod_gzip的模块，但 Apache2.x系列已经内置了Deflate模块，因此，只需要安装Deflate模块即可,由于在编译apache时没有加这些参数，只能另外添加安装：

==========

一、**mod_deflate.so**安装过程：

1、进入源码包

[root@dns filters]# pwd
/home/redhat/httpd-2.2.15/**modules/filters**

2、安装deflate 并添加到httpd.conf中
[root@dns filters]# /usr/local/apache2/bin/apxs -i -c -a mod_deflate.c

apxs命令参数说明：
-i 此选项表示需要执行安装操作，以安装一个或多个动态共享对象到服务器的**modules**目录中。
-a 此选项自动增加一个**LoadModule**行到**httpd.conf**文件中，以激活此模块，或者，如果此行已经存在，则启用之。
-A 与 -a 选项类似，但是它增加的LoadModule命令有一个井号前缀(#)，即此模块已经准备就绪但尚未启用。
-c 此选项表示需要执行编译操作。它首先会编译**C**源程序**(.c)files**为对应的目标代码文件**(.o)**，然后连接这些目标代码和**files**中其余的目标代码文件**(.o**和**.a)**，以生成动态共享对象**dsofile** 。如果没有指定 **-o** 选项，则此输出文件名由**files**中的第一个文件名推测得到，也就是默认为**mod_name.so** 。

3、查看安装目录中的modules文件夹和httpd.conf文件
[root@dns modules]# ls
httpd.exp libphp5.so mod_deflate.so

在第56 行出现
LoadModule deflate_module modules/mod_deflate.so

二、**mod_headers.so**安装过程：

1、mod_headers.c 在源码包的modeules/metadata目录下面

[root@dns metadata]# pwd
/home/redhat/httpd-2.2.15**/modules/metadata**
[root@dns metadata]# ls mod_headers.
mod_headers.c mod_headers.dsp mod_headers.exp mod_headers.lo mod_headers.o

2、安装deflate 并添加到httpd.conf中
[root@dns metadata]# /usr/local/apache2/bin/apxs **-i -a -c** mod_headers.c

[root@dns modules]# ls
httpd.exp libphp5.so mod_deflate.so mod_headers.so

在57行出有
LoadModule headers_module   modules/mod_headers.so

如果是新安装apache，直接加上 --enable-headers --enable-deflate 即可
==========

三、在httpd.conf中添加
<IfModule mod_deflate.c>
  DeflateCompressionLevel 7
  AddOutputFilterByType DEFLATE text/html text/plain text/xml   application/x-httpd-php
   AddOutputFilter DEFLATE js css
</IfModule>

这样的做法可以压缩一般网页中会用到的html、xml、php、css、js等格式文档输出，减少资料传输量，减少网络带宽被吃掉的情形。

DeflateCompressionLevel 指压缩程度的等级；从1到9 ，9是最高等级，据了解，这样做最高可以减少8成大小的传输量，最好也能节省一半；

DeflateCompressionLevel预设可以采用6 这个数值，以维持用处理器效能与网页
压缩品质的平衡
~~~~~~~~~~~~~~

您不得不看 **apache**添加模块（不重新编译）

下面是使用**apxs** 工具给apache添加模块

添加的模块：
LoadModule proxy_module   libexec/mod_proxy.so
LoadModule rewrite_module   libexec/mod_rewrite.so
LoadModule headers_module   libexec/mod_headers.so

添加方法：
1. 进入apache1.3.33 源文件modules目录
cd apache_1.3.33/src/modules/

2. 安装 proxy_module
> cd proxy
> /user/local/apache/bin/apxs -i -a -c *.c

3. 安装 rewrite_module
> cd standard
> /user/local/apache/bin/apxs -i -a -c mod_rewrite.c

4. 安装 headers_module
> cd standard
> /user/local/apache/bin/apxs -i -a -c mod_headers.c

但是，用这种方法添加的proxy不可用，提示错误：
/apache/httpd/bin/apachectl configtest
httpd: Syntax error on line 58 of /apache/httpd-2.2.3/conf/httpd.conf: Cannot load /apache/httpd-2.2.3/modules/mod_proxy.so into server: /apache/httpd-2.2.3/modules/mod_proxy.so: **undefined symbol: proxy_lb_workers**

所以，应该这样：
~~~~~~~~~~~~~~

在**Linux**系统下，需要给已经运行的**Apache**增加**mod_proxy**模块，编译的时候应该这样：

apxs -c -i mod_proxy.c proxy_util.c

否则你可能会收到这样的错误信息

[root@server1 proxy]# /apache/httpd/bin/apachectl configtest
httpd: Syntax error on line 58 of /apache/httpd-2.2.3/conf/httpd.conf: Cannot load /apache/httpd-2.2.3/modules/mod_proxy.so into server: /apache/httpd-2.2.3/modules/mod_proxy.so: **undefined symbol: proxy_lb_workers**

如果你还加载了mod_proxy_ajp.so那应该

apxs -c -i mod_proxy_ajp.c ajp*.c
[root@server1 proxy]# /apache/httpd/bin/apachectl configtest
httpd: Syntax error on line 58 of /apache/httpd-2.2.3/conf/httpd.conf: Cannot load /apache/httpd-2.2.3/modules/mod_proxy_ajp.so into server: /apache/httpd-2.2.3/modules/mod_proxy_ajp.so: **undefined symbol:** ajp_msg_reset

这是为什么呢？请看

apxs - Apache 扩展工具

`apxs` 是一个为Apache HTTP服务器编译和安装扩展模块的工具，用于编译一个或多个源程序或目标代码文件为动态共享对象，使之可以用由`mod_so`提供的`LoadModule` 指令在运行时加载到Apache服务器中。

因此，要使用这个扩展机制，你的平台必须支持DSO特性，而且Apache `httpd` 必须内建了`mod_so` 模块。`apxs` 工具能自动探测是否具备这样的条件，你也可以自己用这个命令手动探测：

```
$ httpd -l
```

该命令的输出列表中应该有`mod_so` 模块。如果所有这些条件均已具备，则可以很容易地借助`apxs` 安装你自己的DSO模块以扩展Apache服务器的功能：

```
$ apxs -i -a -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
cp mod_foo.so /path/to/apache/modules/mod_foo.so
chmod 755 /path/to/apache/modules/mod_foo.so
[activating module 'foo' in /path/to/apache/etc/httpd.conf]
$ apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module foo_module
/path/to/apache/sbin/apachectl restart: httpd started
$ _
```

其中的参数*files* 可以是任何C源程序文件(.c)、目标代码文件(.o)、甚至是一个库(.a)。`apxs` 工具会根据其后缀自动编译C源程序或者连接目标代码和库。但是，使用预编译的目标代码时，必须保证它们是地址独立代码(PIC)，使之能被动态地加载。如果使用GCC编译，则应该使用 `-fpic` 参数；如果使用其他C编译器，则应该查阅其手册，为`apxs` 使用相应的编译参数。

有关Apache对DSO的支持的详细信息，可以阅读`mod_so` 文档，或者直接阅读`src/modules/standard/mod_so.c` 源程序。

语法

apxs **-g** [**-S** *name* =*value*] **-n** *modname*

apxs **-q** [**-S** *name* =*value*] *query* ...

apxs **-c** [**-S** *name* =*value*] [**-o** *dsofile*] [**-I** *incdir*] [**-D** *name* =*value*] [**-L** *libdir*] [**-l** *libname*] [**-Wc,** *compiler-flags*] [**-Wl,** *linker-flags*]*files* ...

apxs **-i** [**-S** *name* =*value*] [**-n** *modname*] [**-a**] [**-A**] *dso-file* ...

apxs -e [**-S** *name =value*] [**-n** *modname*] [**-a**] [**-A**] *dso-file ...*

选项

一般选项

`-n` *modname*

它明确设置了 `-i` (安装)和 `-g` (模板生成)选项的模块名称。对 `-g` 选项，它是必须的；对 `-i` 选项，`apxs` 工具会根据源代码判断，或(在失败的情况下)按文件名推测出这个模块的名称。

查询选项

`-q`

查询某种`apxs` 设置的信息。该选项的*query* 参数可以是下列一个或多个字符串：`CC` ，`CFLAGS` ，`CFLAGS_SHLIB` ，`INCLUDEDIR` ，`LD_SHLIB` ，`LDFLAGS_SHLIB` ，`LIBEXECDIR` ，`LIBS_SHLIB` ，`SBINDIR` ，`SYSCONFDIR` ，`TARGET` 。

这个参数用于手动查询某些设置。比如，要手动处理Apache的C头文件，可以在Makefile中使用：

```
INC=-I`apxs -q INCLUDEDIR`
```

配置选项

`-S` *name =value*

此选项可以改变apxs的上述设置。

模板生成选项

`-g`

此选项生成一个名为*name* 的子目录(见选项 `-n`)和其中的两个文件：一个是名为`mod_`*name* `.c` 的样板模块源程序，可以用来建立你自己的模块，或是学习使用apxs机制的良好开端；另一个则是对应的`Makefile` ，用于编译和安装此模块。

DSO编译选项

`-c`

此选项表示需要执行编译操作。它首先会编译C源程序(.c)*files* 为对应的目标代码文件(.o)，然后连接这些目标代码和*files* 中其余的目标代码文件(.o和.a)，以生成动态共享对象***dsofile*** 。如果没有指定 `-o` 选项，则此输出文件名由*files* 中的第一个文件名推测得到，也就是默认为`mod_`*name* `.so` 。

`-o` *dsofile*

明确指定所建立的动态共享对象的文件名，它不能从*files* 文件列表中推测得到。如果没有明确指定，则其文件名将为`mod_unknown.so` 。

`-D` *name =value*

此选项直接传递到给编译命令，用于增加自定义的编译变量。

`-I` *incdir*

此选项直接传递到给编译命令，用于增加自定义的包含目录。

`-L` *libdir*

此选项直接传递到给连接命令，用于增加自定义的库文件目录。

`-l` *libname*

此选项直接传递到给连接命令，用于增加自定义的库文件。

`-Wc,`*compiler-flags*

此选项用于向编译命令 `libtool --mode=compile` 中附加*compiler-flags* ，以增加编译器特有的选项。

`-Wl,`*linker-flags*

此选项用于向连接命令 `libtool --mode=link` 中附加*linker-flags* ，以增加连接器特有的选项。

DSO的安装和配置选项

-i

此选项表示需要执行安装操作，以安装一个或多个动态共享对象到服务器的*modules* 目录中。

-a

此选项自动增加一个LoadModule 行到httpd.conf 文件中，以激活此模块，或者，如果此行已经存在，则启用之。

-A

与 -a 选项类似，但是它增加的LoadModule 命令有一个井号前缀(#)，即此模块已经准备就绪但尚未启用。

-e

表示需要执行编辑操作，它可以与 -a 和 -A 选项配合使用，与 -i 操作类似，修改Apache的httpd.conf 文件，但是并不安装此模块。

举例

假设有一个扩展Apache功能的模块mod_foo.c ，使用下列命令，可以将C源程序编译为共享模块，以在运行时加载到Apache服务器中：

```
$ apxs -c mod_foo.c
/path/to/libtool --mode=compile gcc ... -c mod_foo.c
/path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
$ _
```

然后，必须修改Apache的配置，以确保有一个LoadModule 指令来加载此共享对象。为了简化这一步骤，apxs 可以自动进行该操作，以安装此共享对象到"modules"目录，并更新httpd.conf 文件，命令如下：

```
$ apxs -i -a mod_foo.la
/path/to/instdso.sh mod_foo.la /path/to/apache/modules
/path/to/libtool --mode=install cp mod_foo.la /path/to/apache/modules ... chmod 755 /path/to/apache/modules/mod_foo.so
[activating module 'foo' in /path/to/apache/conf/httpd.conf]
$ _
```

如果配置文件中尚不存在，会增加下列的行：

```
LoadModule foo_module modules/mod_foo.so
```

如果你希望默认禁用此模块，可以使用 -A 选项，即：

```
$ apxs -i -A mod_foo.c
```

要快速测试apxs机制，可以建立一个Apache模块样板及其对应的Makefile ：

```
$ apxs -g -n foo
Creating [DIR] foo
Creating [FILE] foo/Makefile
Creating [FILE] foo/modules.mk
Creating [FILE] foo/mod_foo.c
Creating [FILE] foo/.deps
$ _
```

然后，立即可以编译此样板模块为共享对象并加载到Apache服务器中：

```
$ cd foo
$ make all reload
apxs -c mod_foo.c
/path/to/libtool --mode=compile gcc ... -c mod_foo.c
/path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
apxs -i -a -n "foo" mod_foo.la
/path/to/instdso.sh mod_foo.la /path/to/apache/modules
/path/to/libtool --mode=install cp mod_foo.la /path/to/apache/modules ... chmod 755 /path/to/apache/modules/mod_foo.so
```

```
[activating module 'foo' in /path/to/apache/conf/httpd.conf]
apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module foo_module
/path/to/apache/sbin/apachectl restart: httpd started
$ _~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

在apache下添加fastcgi模块

1、搭建环境

CentOS5.1，系统只带的apache2.23，默认的apache安装没有带fastcgi模块，要自己手动添加

2、下载

wget http://www.fastcgi.com/dist/mod_fastcgi-2.4.6.tar.gz

3、安装

tar zxf mod_fastcgi-2.4.6.tar.gz
cd mod_fastcgi-2.4.6
apxs -o mod_fastcgi.so -c *.c
apxs -i -a -n fastcgi .libs/mod_fastcgi.so

看看modules是否有mod_fastcgi.so，看看httpd.conf文件里是否有mod_fastcgi.so

再看看httpd.conf里是否有下面这行，没有加上

AddHandler fastcgi-script .fcg .fcgi .fpl

PS：所有都是看源安装目录下的INSTALL文件。

4、apache相关配置

```
<IfModule mod_fastcgi.c>
  FastCgiWrapper /usr/local/apache/bin/suexec
  # URIs that begin with /fcgi-bin/, are found in /var/www/fcgi-bin/
  Alias /fcgi-bin/ /var/www/fcgi-bin/
  ScriptAlias /wws /var/www/fcgi-bin/b.fcgi
  # Anything in here is handled as a "dynamic" server if not defined as "static" or "external"
  <Directory /var/www/fcgi-bin/>
        AllowOverride None
        Options +ExecCGI -Includes
        #SetHandler fastcgi-script
        AddHandler fastcgi-script .fcg .fcgi
        Order allow,deny
        Allow from all
  </Directory>
  # Anything with one of these extensions is handled as a "dynamic" server if not defined as
  # "static" or "external". Note: "dynamic" servers require ExecCGI to be on in their directory.
  #AddHandler fastcgi-script .fcgi .fpl
  # Start a "static" server at httpd initialization inside the scope of the SetHandler
  #FastCgiServer /var/www/fcgi-bin/echo -processes 3
  # Start a "static" server at httpd initialization inside the scope of the AddHandler
  #FastCgiServer /var/www/fcgi-bin/b.fcgi -processes 3 -user nobody -group nobody
  #FastCgiServer /var/www/htdocs/some/path/echo.fcgi
  # Start a "static" server at httpd initialization outside the scope of the Set/AddHandler
  #FastCgiServer /var/www/htdocs/some/path/coolapp
```

```
  #<Directory /var/www/htdocs/some/path/coolapp>
  # SetHandler fastcgi-script
  #</Directory>
</IfModule>
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

为**Apache**编译添加**mod_expires**模块出错的解决笔记**[**原创**]**

注：/usr/local/apache/为Apache的安装路径，/opt/httpd-2.0.55/为Apache的源代码目录。

Linux下，执行以下命令为Apache添加mod_expires模块：

```
/usr/local/apache/bin/apxs -i -a -c /opt/httpd-2.0.55/modules/metadata/mod_expires.c
```

报错：

引用

Warning! dlname not found in /usr/local/apache/modules/mod_expires.la.
Assuming installing a .so rather than a libtool archive.
chmod 755 /usr/local/apache/modules/mod_expires.so
chmod: 无法访问‘/usr/local/apache/modules/mod_expires.so’: 没有那个文件或目录
apxs:Error: Command failed with rc=65536

再执行：

```
gcc -shared -o /usr/local/apache/modules/mod_expires.so /opt/httpd-
2.0.55/modules/metadata/mod_expires.o
/usr/local/apache/bin/apxs -i -a -c /opt/httpd-2.0.55/modules/metadata/mod_expires.c
```

编译成功。
##################################

这里我来解释下什么叫动态加载和静态加载？

说简单，并用打比喻的方式来解释.

好比有两个人a和m
a代表apache,m代表module
要想让a使用m的东西
一个方法是把m的东西都放到a那里去，a使用的时候就是现成的了
就是所谓的静态编译

还有一个方法，
就是告诉a， m的住址，当a要使用m的东西的时候,a去找m,然后使用
不过，这种方法要注意的一个问题就是：m必须要有实际的住址，
否则a会找不到m而产生错误的,我此文开始提到的 apachectl startssl产生
的错误就是这个原因，应该再编译好ssl才可以的.
这种方法也就是apache 的动态(DSO)编译了

静态：
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

在使用./configure 编译的时候，如果不指定某个模块为动态，即没有使用：enable-mods-shared=module或者enable-module=shared 这个2个中的一个，那么所有的默认模块为静态。 那么何谓静态？ 其实就是编译的时候所有的模块自己编译进 httpd 这个文件中（我们启动可以使用这个执行文件,如： ./httpd & ） ，启动的时候这些模块就已经加载进来了，也就是可以使用了， 通常为：<ifmodule> </ifmodule> 来配置。所以大家看到的配置都是 <ifmodule module.c> ，很显然，module.c这个东西已经存在 httpd这个文件中了。

动态：

就是编译的时候，使用enable-module=shared 或者enable-modules-shared=module 来动态编译。 那么什么是动态？ 静态是直接编译进httpd中， 那么动态显然就不编译进去了，也就是你启动的时候根本不会加载这个模块， 而是给你一个module.so 文件，你一定要使用 loadmodule 这个语法来加载，这个模块才有效。

那么区别就出来了：静态的模块通常是<ifmodule></ifmodule> 来配置， 动态使用loadmoule来加载，然后再配置。
首先看看编译apache的选项含义

对于apache 1.3.x
./configure --prefix=/usr/local/apache \
--enable-module=so \
--enable-module=most \
--enable-shared=max \
--enable-module=rewrite

对于apache 2.0.x
./configure --prefix=/usr/local/apache2 \
--enable-modules=most \
--enable-mods-shared=all \
--enable-so \
--enable-rewrite

对于apache 2.2.0
./configure --prefix=/usr/local/apache2 \
--enable-mods-shared=all \
--enable-so \
--enable-rewrite

举例一:编译一个apache2.2.8版本
./configure --prefix=/usr/local/apache --enable-so --enable-mods-shared=most --enable-rewrite --enable-forward
说明:
so模块用来提供 DSO 支持的 apache 核心模块.

--enable-so 选项：让 Apache 可以支持DSO模式，注意，这里采用的是 Apache2.0 的语法。如果你的Apache 是1.3版本，应改为--enable-module=so

--enable-mods-shared=most选项：

告诉编译器将所有标准模块都动态编译为DSO模块。
如果用的是 Apache1.3, 改为--enable-shared=max就可以。

-enable-rewrite选项：支持地址重写功能，使用1.3版本的朋友请将它改为--enable-module=rewrite

--enable-module=most
用most可以将一些不常用的，不在缺省常用模块中的模块编译进来.
--enable-mods-shared=all意思是动态加载所有模块，如果去掉-shared话，是静态加载所有模块.

举例二:
执行 ./configure --prefix=/server/apache/ \
--enable-deflate=shared \
--enable-headers=shared \
--enable-rewrite=shared \
--enable-mods-shared=most

--enable-mods-shared=all意思是动态加载所有模块，如果去掉-shared话，是静态加载所有模块。
--enable-mods-shared=most则是动态编译大部分常用的模块，当然,也可以有选择的加载一些模块，most意思是只包含通常用的模块，并且以动态加载模式加载. 记住apache1.xx和apache2.xx的模块编译写法是不一样的.
如已经安装好，到modules目录下，查看里面是否有一些.so文件，以此为扩展名的文件为模块文件。表明系统已经加载了模块，并且是动态方式加载的。如果modules目录下没有这些.so文件，表明系统没有动态加载模块。 然后我们就点看看系统有没有通过静态方式加载一些模块，通过命令：apachectl -l 会列出系统已经加载的模块，且为静态方式加载的模块。

4。静态加载的模块不会显示在modules目录下，静态加载模块内嵌在系统里，如果想卸载该模块，Apache需要重新进行编译安装。 静态加载的方法是配置时指定启动哪些模块，语法是：./configure --enable-deflate --prefix=/opt/http2 make make install ./bin/apachectl restart

5。动态加载的模块都会显示在modules目录下，要想让这些模块起作用，还需要在主配置文件里装载目录下的这些模块，语法是：LoadModule deflate_module modules/mod_deflate.so

6。模块加载后，要想让模块起作用，需要在配置文件里添加相应的配置信息，具体配置信息可参考Apache手册中关于模块配置部分。

7。要想让模块起作用，无论是通过静态方式加载的模块，还是通过动态方式加载的模块，都需要在配置文件里添加相应配置信息。 区别是如下语法，动态加载的模块，在配置文件中只需要输入<IfModule >.....</IfModule>里面的内容，但要是静态加载的模块，在配置文件中要保那对标记也写进配置文件。

<IfModule prefork.c>
StartServers 8
MinSpareServers 5
MaxSpareServers 20
ServerLimit 256
MaxClients 256
MaxRequestsPerChild 4000
</IfModule>

8。 配置信息可以直接在主配置文件（httpd.conf)中进行添加，也可以在子配置文件中进行添加；不同的是如果配置信息写在子配置文件中，在主配置文件里还需要加一条语句，声明主配置文件包括子配置信息，语法是：Include conf.d/*.conf (声明主配置文件包括conf.d目录中的所有子配置文件里的信息)

9。 重启Apache服务， 语法是./bin/apachectl restart

httpd 命令

httpd -M 用来列出基于当前配置加载的所有模块

httpd -l 输出一个静态编译在服务器中的模块的列表。它不会列出使用LoadModule指令动态加载的模块
httpd -S 显示虚拟机的设置

httpd -t 对配置文件执行语法检查
httpd -v 显示httpd的版本
httpd -V 显示httpd的版本和编译参数
httpd -f 在启动中使用的配置文件
httpd -e levle 在服务器启动时，设置LogLevel为level 。它用于在启动时，临时增加出错信息的详细程度，以帮助排错

apachectl 命令
apachectl脚本有两种操作模式
1. 启动httpd ，并传递所有的命令行参数。
2. 作为SysV初始化脚本，接受简单的一个单词的参数，如：start, restart, stop ，并把他们翻译为适当的信号发送给httpap
apachectl start 启动apache httpd后台守护进程
apachectl stop 停止apache httpd后台守护进程
apachectl restart 重启apache httpd 后台守护进程
apachectl fullstatus 显示由mod_status提供的完整的状态报告.要使用这个功能，需要启用服务器上的mod_status模块
apachectl startssl 以支持SSL的方式启动httpd 需要ssl模块

apxs 命令
apxs是一个为Apache HTTP服务器编译和安装扩展模块的工具,用于编译一个或多个源程序或目标代码文件为动态共享对象
apxs -i -a -c mod_foo.c 将mod_foo.c 编译成共享模块
htpasswd命令
htpasswd建立和更新用于基本认证的存储用户名/密码的文本文件
htpasswd /usr/local/nagios/etc/htpasswd.users jsmith 添加或修改用户jsmith的密码
htpasswd -c /usr/local/nagios/etc/htpasswd.users jane 创建一个新文件并在其中添加一条用户jane的记录

```
AuthName "Access"
AuthType Basic
AuthUserFile /usr/local/nagios/etc/htpasswd.users
Require valid-user
```

下面是使用**apxs**工具给apache添加模块

添加的模块:

```
LoadModule proxy_module      libexec/mod_proxy.so
LoadModule rewrite_module    libexec/mod_rewrite.so
LoadModule headers_module    libexec/mod_headers.so
```

添加方法:
1. 进入apache 源文件modules目录
cd apache/src/modules/

2. 安装 proxy_module
> cd proxy
> /user/local/apache/bin/**apxs -i -a -c *.c**

3. 安装 rewrite_module
> cd standard
> /user/local/apache/bin/**apxs -i -a -c mod_rewrite.c**

4. 安装 headers_module

> cd standard
> /user/local/apache/bin/**apxs -i -a -c mod_headers.c**

但是，用这种方法添加的proxy不可用，提示错误：

/apache/httpd/bin/**apachectl configtest**

httpd: Syntax error on line 58 of /apache/httpd-2.2.3/conf/httpd.conf: Cannot load /apache/httpd-2.2.3/modules/mod_proxy.so into server: /apache/httpd-2.2.3/modules/mod_proxy.so: undefined symbol: proxy_lb_workers

所以，应该这样：

在**Linux**系统下，需要给已经运行的**Apache**增加**mod_proxy**模块，编译的时候应该这样：

apxs -c -i mod_proxy.c proxy_util.c

否则你可能会收到这样的错误信息

[root@server1 proxy]# /apache/httpd/bin/apachectl configtest
httpd: Syntax error on line 58 of /apache/httpd-2.2.3/conf/httpd.conf: Cannot load /apache/httpd-2.2.3/modules/mod_proxy.so into server: /apache/httpd-2.2.3/modules/mod_proxy.so: undefined symbol: proxy_lb_workers

加载后重新启动Apache
[root@server1 proxy]# /apache/httpd/bin/**apachectl start**

这样我可就可以重用重写和代理功能了

关闭时：

/user/local/apache/bin/**apxs -i -a -c mod_rewrite.c**

启动时：

/user/local/apache/bin/**apxs -c -i mod_rewrite.c**

---

mod_ssl报错，装了最新的apache

今天回来发现**apache**不能启动了，郁闷，报的错误是**mod_ssl**无法加载，我把**#LoadModule ssl_module modules/mod_ssl.so**加截**ssl**的地方注释掉就好了。以前都配置好了的，今天怎么会不行了呢？我又重新把**mod_ssl**编译了一下，

cd /apache的解压目录／modules/ssl

[root@BlackGhost ssl]# /usr/local/apache/bin/apxs -c -i mod_ssl.c
ssl_private.h:541:14: error: declaration for parameter 'ssl_hook_UserCheck' but no such parameter
ssl_private.h:540:14: error: declaration for parameter 'ssl_hook_Auth' but no such parameter
ssl_private.h:537:14: error: declaration for parameter 'ssl_init_ModuleKill' but no such parameter
ssl_private.h:536:14: error: declaration for parameter 'ssl_init_Child' but no such parameter
mod_ssl.c:520:1: error: expected '{' at end of input
apxs:Error: Command failed with rc=65536

上面的报错只是最下面的一部分，搞了好长时间，没搞定，决定下个新的apache重装一下，顺便把安装过程说一下

一，安装**httpd-2.2.16.tar.gz**

wget http://www.apache.org/dist/httpd/httpd-2.2.16.tar.gz

tar zxvf httpd-2.2.16.tar.gz

cd httpd-2.2.16

./configure –prefix=/usr/local/apache –enable-ssl –enable-so –with-ssl=/us

make && make install

1，改**documentroot**和**<Directory “/home/zhangy/www”>**，把里面的默认路径改成你的**web**目录。

2，加载**libphp.so**

我记得以前装php的时候，会自动添加好下面的东西，但是这次我只升级apache，我还没找到libphp.c文件在什么地方，没法用apxs来重新编译，还是用以前的libphp5.so吧

LoadModule php5_module modules/libphp5.so
AddType application/x-httpd-php .php .phtml
AddType application/x-httpd-php-source .phps
AddType application/x-httpd-php .php

如果不加上面的东西，php代码无法解悉，会直接把源码显示出来

3，修改**user**和**group**，你看一下**user**后面的用户，在你的系统中有没有，如果没有的话，你启动**apache**后，查看页面时会**forbidden**错误，改成你指定的用户，当然这个用户必有在你的系统中。

User/Group: The name (or #number) of the user/group to run httpd as.
It is usually good practice to create a dedicated user and group for
running httpd, as with most system services.
#
User zhangy
Group users

二，编译**mod_ssl**模块看看报不报错

cd /apache的解压目录／modules/ssl

/usr/local/apache/bin/apxs -c -i mod_ssl.c

make && make install

下面是安装时候显示出来的信息

Libraries have been installed in:
/usr/local/apache/modules

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
- add LIBDIR to the `LD_LIBRARY_PATH' environment variable
during execution
- add LIBDIR to the `LD_RUN_PATH' environment variable
during linking
- use the `-Wl,–rpath -Wl,LIBDIR' linker flag
- have your system administrator add LIBDIR to `/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
———————————————————————-

chmod 755 /usr/local/apache/modules/mod_ssl.so

现在编译就没有报错了，我觉得肯定是因为我升级的原因，并且只是升级了一部分。安装的时候，我已经内建了mod_ssl所以就不用在加载了。如果加了的话，会提示**module ssl_module is built-in and can't be loaded**

[root@BlackGhost misc]# /usr/local/apache/bin/apachectl configtest
Syntax OK

对**apache2**的模块如果动态编译通常可以使用 **/path/apxs -c *.c**来完成

但对 **mod_ssl**编译是会有一些问题

如:
出现
Unrecognized SSL Toolkit!
是由于 HAVE_OPENSSL这个没有define
需要增加 -DHAVE_OPENSSL
undefined symbol: ssl_cmd_SSLMutex
或
undefined symbol: X509_free
通产是由于静态连接了 openssl的库照成的(默认）。
需要使用 -lcrypto -lssl -ldl
命令如下:

/path/apxs -I/path/openssl/include -L/path/openssl/lib -c *.c -l crypto -l ssl -l dl

openssl 编译的时候需要增加 shared参数

/usr/local/apache2/bin/apxs -a -i -c -D HAVE_OPENSSL=1 -I /usr/include/openssl/ -L /usr/lib/openssl/engines/lib ./mod_ssl.c

---

Apache 编译参数的区别

1、 ./configure --prefix=/usr/local/apache
其中，下面这两个参数加不加都是一样的。

--enable-so、--enable-so=static

[root@haha bin]# ./apachectl -M
Loaded Modules:
core_module (static)
authn_file_module (static)
authn_default_module (static)
authz_host_module (static)
authz_groupfile_module (static)
authz_user_module (static)
authz_default_module (static)
auth_basic_module (static)
include_module (static)
filter_module (static)
log_config_module (static)
env_module (static)
setenvif_module (static)
mpm_prefork_module (static)
http_module (static)
mime_module (static)
status_module (static)
autoindex_module (static)
asis_module (static)
cgi_module (static)
negotiation_module (static)
dir_module (static)
actions_module (static)
userdir_module (static)

alias_module (static)
so_module (static)
Syntax OK

2、./configure --prefix=/usr/local/apache --enable-mods-shared=most（编译大部分常用模块，也可以是all，即编译所有模块。）
[root@haha apache]# bin/apachectl -M
Loaded Modules:
core_module (static)
mpm_prefork_module (static)
http_module (static)
so_module (static)
---------- 下面是动态加载的模块
authn_file_module (shared)
authn_dbm_module (shared)
authn_anon_module (shared)
authn_dbd_module (shared)
authn_default_module (shared)
authz_host_module (shared)
authz_groupfile_module (shared)
authz_user_module (shared)
authz_dbm_module (shared)
authz_owner_module (shared)
authz_default_module (shared)
auth_basic_module (shared)
auth_digest_module (shared)
dbd_module (shared)
dumpio_module (shared)
ext_filter_module (shared)
include_module (shared)
filter_module (shared)
substitute_module (shared)
deflate_module (shared)
log_config_module (shared)
logio_module (shared)
env_module (shared)
expires_module (shared)
headers_module (shared)
ident_module (shared)
setenvif_module (shared)
mime_module (shared)
dav_module (shared)
status_module (shared)
autoindex_module (shared)
asis_module (shared)
info_module (shared)
cgi_module (shared)
dav_fs_module (shared)
vhost_alias_module (shared)
negotiation_module (shared)
dir_module (shared)

imagemap_module (shared)
actions_module (shared)
speling_module (shared)
userdir_module (shared)
alias_module (shared)
rewrite_module (shared)