**[apache 模块开发之输出过滤器](http://andrew913.iteye.com/blog/399025)**

**博客分类：**

* [apache 模块开发](http://andrew913.iteye.com/category/67192)

[Apache](http://www.iteye.com/blogs/tag/Apache)[Struts](http://www.iteye.com/blogs/tag/Struts)[Tomcat](http://www.iteye.com/blogs/tag/Tomcat)[C](http://www.iteye.com/blogs/tag/C)[C++](http://www.iteye.com/blogs/tag/C++)

前面写了apache模块开发之hellloworld ,那个例子是一个简单的内容生成器。现在我们来讲一个过滤器的实现。过滤器是apache中一个非常精妙的设计，apache自带的很多模块都是通过过滤器来实现。   
  
对于过滤器，有输入过滤器与输出过滤器两种。   
对于输入过滤器，就是在内容生成器之前执行，而输出过滤器则在内容生成器之后。可以有下面的顺序：   
http请求-》输入过滤器-》内容生成-》输出过滤器-》用户   
  
所有的请求都会经过我们的过滤器，所以我们可以对这些进行操作，比如统计流量，压缩等等。   
  
下面我们讲一个例子，这个例子是apache自带的一个demo，它的功能是把页面中所有的小写字母变成大写字母。  
从上面的过滤器流程中我们可以找到，只要我们在输出过滤器中挂一个钩子，然后将所有的字符变成大写就OK了。   
代码如下：

**C代码  [收藏代码](javascript:void())**

1. #include "httpd.h"
2. #include "http\_config.h"
3. #include "apr\_buckets.h"
4. #include "apr\_general.h"
5. #include "apr\_lib.h"
6. #include "util\_filter.h"
7. #include "http\_request.h"
9. #include <ctype.h>
11. **static** **const** **char** s\_szCaseFilterName[]="CaseFilter";
12. module AP\_MODULE\_DECLARE\_DATA case\_filter\_module;
14. **typedef** **struct**
15. {
16. **int** bEnabled;
17. } CaseFilterConfig;
19. **static** **void** \*CaseFilterCreateServerConfig(apr\_pool\_t \*p,server\_rec \*s)
20. {
21. CaseFilterConfig \*pConfig=apr\_pcalloc(p,**sizeof** \*pConfig);
23. pConfig->bEnabled=0;
25. **return** pConfig;
26. }
28. **static** **void** CaseFilterInsertFilter(request\_rec \*r)
29. {
30. CaseFilterConfig \*pConfig=ap\_get\_module\_config(r->server->module\_config,
31. &case\_filter\_module);
33. **if** (!pConfig->bEnabled)
34. **return**;
36. ap\_add\_output\_filter(s\_szCaseFilterName,NULL,r,r->connection);
37. }
39. **static** apr\_status\_t CaseFilterOutFilter(ap\_filter\_t \*f,
40. apr\_bucket\_brigade \*pbbIn)
41. {
42. request\_rec \*r = f->r;
43. conn\_rec \*c = r->connection;
44. apr\_bucket \*pbktIn;
45. apr\_bucket\_brigade \*pbbOut;
47. pbbOut=apr\_brigade\_create(r->pool, c->bucket\_alloc);
48. **for** (pbktIn = APR\_BRIGADE\_FIRST(pbbIn);
49. pbktIn != APR\_BRIGADE\_SENTINEL(pbbIn);
50. pbktIn = APR\_BUCKET\_NEXT(pbktIn))
51. {
52. **const** **char** \*data;
53. apr\_size\_t len;
54. **char** \*buf;
55. apr\_size\_t n;
56. apr\_bucket \*pbktOut;
58. **if** (APR\_BUCKET\_IS\_EOS(pbktIn))
59. {
60. apr\_bucket \*pbktEOS=apr\_bucket\_eos\_create(c->bucket\_alloc);
61. APR\_BRIGADE\_INSERT\_TAIL(pbbOut,pbktEOS);
62. **continue**;
63. }
65. /\* read \*/
66. apr\_bucket\_read(pbktIn,&data,&len,APR\_BLOCK\_READ);
68. /\* write \*/
69. buf = apr\_bucket\_alloc(len, c->bucket\_alloc);
70. **for** (n=0 ; n < len ; ++n)
71. buf[n] = apr\_toupper(data[n]);
73. pbktOut = apr\_bucket\_heap\_create(buf, len, apr\_bucket\_free,
74. c->bucket\_alloc);
75. APR\_BRIGADE\_INSERT\_TAIL(pbbOut,pbktOut);
76. }
77. apr\_brigade\_cleanup(pbbIn);
78. **return** ap\_pass\_brigade(f->next,pbbOut);
79. }
81. **static** **const** **char** \*CaseFilterEnable(cmd\_parms \*cmd, **void** \*dummy, **int** arg)
82. {
83. CaseFilterConfig \*pConfig=ap\_get\_module\_config(cmd->server->module\_config,
84. &case\_filter\_module);
85. pConfig->bEnabled=arg;
87. **return** NULL;
88. }
90. **static** **const** command\_rec CaseFilterCmds[] =
91. {
92. AP\_INIT\_FLAG("CaseFilter", CaseFilterEnable, NULL, RSRC\_CONF,
93. "Run a case filter on this host"),
94. { NULL }
95. };
97. **static** **void** CaseFilterRegisterHooks(apr\_pool\_t \*p)
98. {
99. ap\_hook\_insert\_filter(CaseFilterInsertFilter,NULL,NULL,APR\_HOOK\_MIDDLE);
100. ap\_register\_output\_filter(s\_szCaseFilterName,CaseFilterOutFilter,NULL,
101. AP\_FTYPE\_RESOURCE);
102. }
104. module AP\_MODULE\_DECLARE\_DATA case\_filter\_module =
105. {
106. STANDARD20\_MODULE\_STUFF,
107. NULL,
108. NULL,
109. CaseFilterCreateServerConfig,
110. NULL,
111. CaseFilterCmds,
112. CaseFilterRegisterHooks
113. };

首先所有程序的入口还是case\_filter\_module这个变量，它和之前的helloworld多了几个选项，其中CaseFilterCreateServerConfig这是一个服务器初始化函数，这个初始化会在apache启动的时候执行。   
我们来看下具体的实现：

**C代码  [收藏代码](javascript:void())**

1. **static** **void** \*CaseFilterCreateServerConfig(apr\_pool\_t \*p,server\_rec \*s)
2. {
3. CaseFilterConfig \*pConfig=apr\_pcalloc(p,**sizeof** \*pConfig);
4. pConfig->bEnabled=0;
5. **return** pConfig;
6. }

这里我们需要讲一下在apache中申请内存空间和普通的c语言的malloc有所区别，apache的申请空间都是向池申请，然后在这个池消失的时候就自动的释放我们的空间。其效率也要高于我们普通c语言malloc的堆空间分配的。   
具体语法：

**C代码  [收藏代码](javascript:void())**

1. CaseFilterConfig \*pConfig=apr\_pcalloc(p,**sizeof** \*pConfig);

这里就是向p这个池申请了空间，这个变量初始化以后，它的生命周期是等同于整个httpd进程的。所以在我们后面的程序中都可以调用。   
  
case\_filter\_module这个变量还有CaseFilterCmds这个变量，这是一个配置命令，也就是说当我们想把一些参数通过配置httpd.conf来传入apache的时候，我们就通过这个函数指针。   
它的结构为：

**C代码  [收藏代码](javascript:void())**

1. **static** **const** command\_rec CaseFilterCmds[] =
2. {
3. AP\_INIT\_FLAG("CaseFilter", CaseFilterEnable, NULL, RSRC\_CONF,
4. "Run a case filter on this host"),
5. { NULL }
6. };

变量第一个参数为我们配置的选项，也就是我们的key，第二个参数是这个配置命令将执行的函数，我们就是通过这个函数来传递我们的配置的。例如本例子我们需要在httpd.conf中加入CaseFilter on这个命令，我们也可以看到CaseFilterEnable的实现为：

**C代码  [收藏代码](javascript:void())**

1. **static** **const** **char** \*CaseFilterEnable(cmd\_parms \*cmd, **void** \*dummy, **int** arg)
2. {
3. CaseFilterConfig \*pConfig=ap\_get\_module\_config(cmd->server->module\_config,
4. &case\_filter\_module);//读配置
5. pConfig->bEnabled=arg;//传递配置
6. **return** NULL;
7. }

case\_filter\_module最后一个参数就是我们的注册函数，具体实现如下：

**C代码  [收藏代码](javascript:void())**

1. **static** **void** CaseFilterRegisterHooks(apr\_pool\_t \*p)
2. {
3. ap\_hook\_insert\_filter(CaseFilterInsertFilter,NULL,NULL,APR\_HOOK\_MIDDLE);
4. ap\_register\_output\_filter(s\_szCaseFilterName,CaseFilterOutFilter,NULL,
5. AP\_FTYPE\_RESOURCE);
6. }

ap\_hook\_insert\_filter 就是通过CaseFilterInsertFilter来讲我们的过滤器加入到apache过滤链中，然后就是ap\_register\_output\_filter这个函数将我们的过滤器模块注册起来。   
当我们需要输出一个页面的时候，就会触发ap\_register\_output\_filter中低二个参数，这个参数为我们的处理函数指针，也就是我们所有的操作都是可以在这里搞定。对于这里，我们把所有的小写字母转化为大写字母，当然我们可以做更多的事情。   
  
最后我们在我们的httpd.conf中加入   
  
Loadmodule\*\*\*\*\*\*//加载我们的模块   
  
再加上我们的配置命令   
CaseFilter on  //开启我们的过滤模块，我们也可以加入很多类似的配置。   
  
编译我们的模块，重新启动apache，在看看我们的apache中所有的页面中的小写变成了大写。   
  
最后还需要讲一点的时候，上面用的很多函数和标准C是有区别的，都是调用的apr的库。所以我们还要去熟悉下这里的函数，总体来说和标准C是差不多的，只是做了一些优化。