HTTP 解析器（fasterhttp）

厉华

版本修订

修订日期	修订版本	修订人	修订内容
2016-07-10	1.0.0	厉华	第一版

目录索引

1 简介

fasterhttp 是一个 C 语言编写的高性能、跨平台、流式 HTTP 协议解析器，实现了通讯数据层和 HTTP 协议解析层的处理，包括 HTTP 请求和响应的数据解析和数据组织。

fasterhttp 对解析的内容不做提取，仅仅指明内容的位置，所以性能非常快。（参见性能压测章节）

fasterhttp 提供了多层次 API 供应用选择使用，可以直接使用高层函数实现单次 HTTP 交互，也可以使用低层函数组合出自定义的 HTTP 解析过程，这在多路复用并发模型中尤其需要。

特性：

支持完整 HTTP/1.x 协议标准

支持 TLS（基于 OpenSSL）

支持同步堵塞、非堵塞流式处理（可以和 select、epoll 配合使用）

可用于 HTTP 客户端/服务端

内置 gzip,deflate 压缩解压处理（基于 zlib）

不依赖其它第三方库（除 OpenSSL 和 zlib 外）

只有一对.h .c，小巧轻量、便于修改、移植和嵌入到项目中

跨平台，支持 UNIX、Linux、WINDOWS

解析器在解析 HTTP 头部时，实现了对下列头部选项的语义检查和处理：

请求方法

HTTP 版本

返回的 HTTP 响应码

Content-Length

Transfer-Encoding: chunked 以及 TRAILER

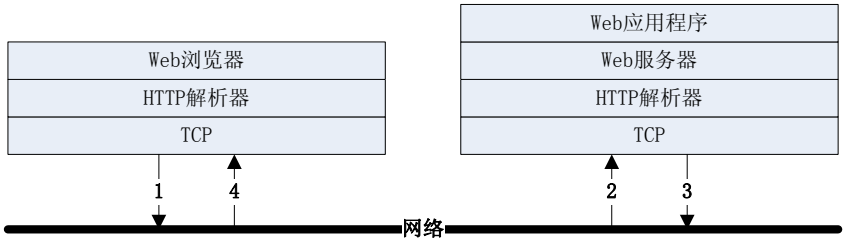
Content-Encoding: gzip,deflate 和 Accept-Encoding: gzip,deflate（可选）

Connection: Keep-Alive 和 Connection: Close

解析器还指明了其它 HTTP 头选项和 HTTP 消息主体位置供直接访问。

2 HTTP 协议简述

HTTP 协议基于 TCP 协议之上。



客户端（Web 浏览器）创建 TCP（也支持 UDP，但很少用）、连接上服务端（Web 服务器），发送 HTTP 请求，服务端接收、解析、处理请求，组织、发送 HTTP 响应回客户端。

如果双方协调保持连接，那么在同一条 TCP 连接上可迭代多轮 HTTP 请求和响应。

2.1 HTTP 报文格式

2.1.1 HTTP 请求格式

HTTP 请求由请求首行、头选项区域、头体分割空行、体数据（可选）四部分组成。完整格式为：

```
请求方法 统一资源标识符 HTTP 版本号<CR><LF>
头选项名: 头选项值<CR><LF>
头选项 2 名: 头选项 2 值<CR><LF>
头选项 3 名: 头选项 3 值<CR><LF>
<CR><LF>
体数据
```

请求首行格式为（三段之间用空格或 TAB 分隔）：

```
请求方法 统一资源标识符 HTTP 版本号<CR><LF>
```

“请求方法”常用的有“GET”（不带体数据的请求）、“POST”（带体数据的请求）等，但不绝对

“统一资源标识符”URI 即统一资源定位符 URL 去掉左边的“协议名”、“域名”或“IP”等信息后的右边虚拟根路径部分，如 HTTP 网址“http://zhidao.baidu.com/question/68016373.html”的 URI 为“/question/68016373.html”

“HTTP 版本号”常用的有“HTTP/1.1”和“HTTP/1.0”，外还有“HTTP/0.9”很少看到了，现在正在制订“HTTP/2.0”标准，不过改动很大了

“<CR><LF>”为换行符，由 ASCII 字符 0x0D 和 0x0A 组成。

头选项区域由 0~n 行头选项行组成，每行格式为：

```
头选项名: 头选项值<CR><LF>
```

其中头选项“Content-Length”的值指明了体数据的长度。如果没有体数据，则无需设置头选项“Content-Length”或设置值为 0。

头选项区域和体数据之间用一个空行分隔

```
<CR><LF>
```

2.1.2 HTTP 响应格式

HTTP 响应由响应首行、头选项区域、头体分割空行、体数据（可选）四部分组成。完整格式为：

```
HTTP 版本号 响应状态码 状态码描述文本<CR><LF>
头选项名: 头选项值<CR><LF>
头选项 2 名: 头选项 2 值<CR><LF>
头选项 3 名: 头选项 3 值<CR><LF>
<CR><LF>
体数据
```

响应首行格式为（三段之间用空格或 TAB 分隔）：

```
HTTP 版本号 响应状态码 状态码描述文本<CR><LF>
```

“响应状态码”最常用的有“200”表示请求成功、“404”表示没有找到页面/文件、“505”服务器内部错误等。

“状态码描述文本”是对“响应状态码”的英文描述，如“200”的描述文本是“OK”、“404”的描述文本是“Not Found”、“505”的描述文本是“Internal

Server Error”等。可以自定义该描述文本。

2.2 HTTP 解析器

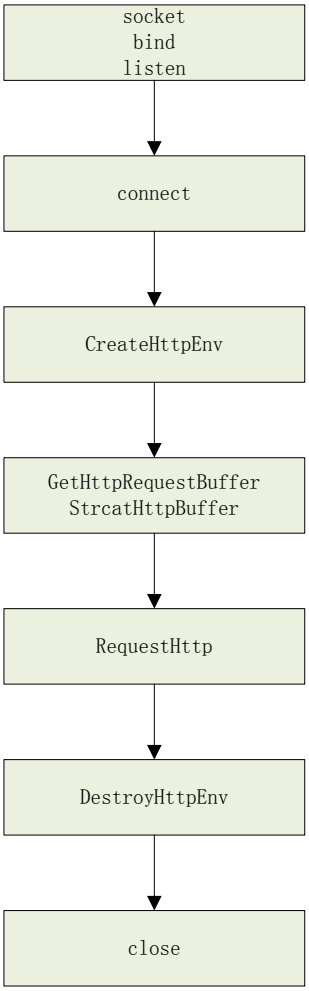
HTTP 解析器的主要功能是读取 TCP 连接上的数据，解析 HTTP 请求或响应，分解出首行、头选项和值、体数据，交由 Web 服务器使用，Web 服务器才能正确对外提供服务，HTTP 解析器的解析性能直接影响 Web 服务器对外提供服务的效率。

3 使用说明

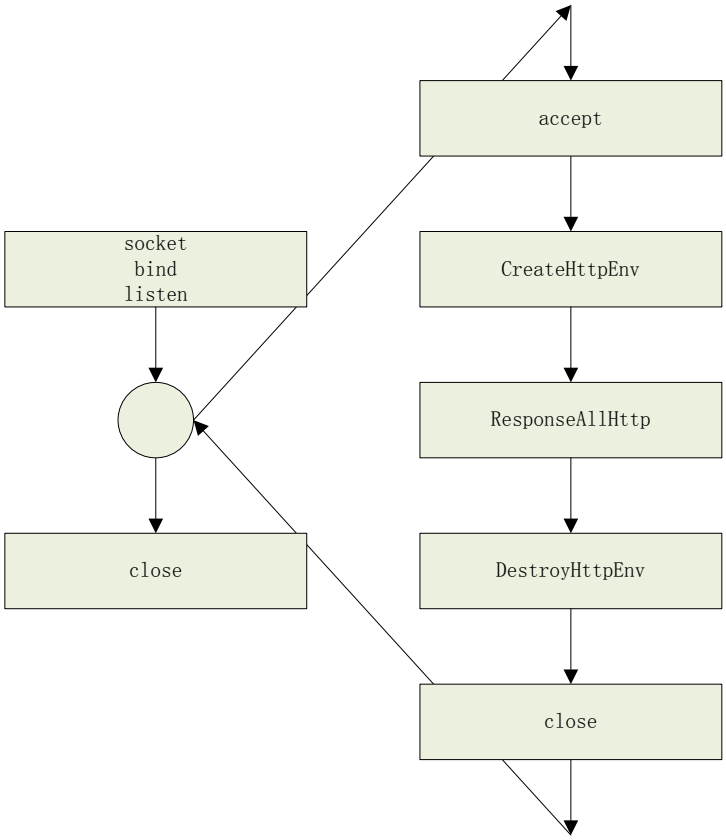
3.1 堵塞模型

每个 HTTP 客户端或服务端使用一个 HTTP 环境对象，调用函数 CreateHttpEnv 来构建

```
struct HttpEnv            *e = NULL ;
…
e = CreateHttpEnv();
…
```



客户端调用 socket 和 connect，调用函数 GetHttpRequestBuffer 和函数 StrcatHttpBuffer 组织 HTTP 请求数据（包括 HTTP 头和 HTTP 体），接着调用函数 RequestHttp 发起单次 HTTP 请求并接收解析 HTTP 响应，调用函数 GetHttpHeaderPtr 得到 HTTP 响应头选项，调用函数 GetHttpBodyPtr 和 GetHttpBodyLen 访问 HTTP 响应体。



服务端调用 socket、bind、listen、accept 建立好一条 TCP 连接，调用函数 ResponseAllHttp 并指定回调函数 ProcessHttpRequest 以处理单次 HTTP 请求、生成 HTTP 响应。回调函数里可以调用函数 GetHttpHeaderPtr 得到 HTTP 请求头选项，调用函数 GetHttpBodyPtr 和 GetHttpBodyLen 访问 HTTP 请求体，调用函数 GetHttpResponseBuffer 和 StrcatHttpBuffer 组织 HTTP 响应数据。

每个 HTTP 环境对象使用完后调用函数 DestroyHttpEnv 销毁之。如需重复使用则调用函数 ResetHttpEnv 重置内部状态。

```
ResetHttpEnv( e );
...
DestroyHttpEnv( e );
```

下面是一对简单的客户端和服务端示例（为突出显示主干，删除了所有出错处理；fasterhttp API 加粗显示），客户端先（位于源代码包 test/test_client_block.c）：

```
#include "fasterhttp.h"

static int TestParseHttpRequest( struct HttpEnv *e , char *str )
{
	SOCKET			connect_sock ;
```

```
	struct sockaddr_in	connect_addr ;

	struct HttpBuffer *b = NULL ;

	int			nret = 0 ;

	ResetHttpEnv( e );

	connect_sock = socket( AF_INET , SOCK_STREAM , IPPROTO_TCP ) ;

	memset( & connect_addr , 0x00 , sizeof(struct sockaddr_in) );
	connect_addr.sin_family = AF_INET;
	connect_addr.sin_addr.s_addr = inet_addr( "127.0.0.1" );
	connect_addr.sin_port = htons( (unsigned short)9527 );

	nret = connect( connect_sock , (struct sockaddr *) & connect_addr , sizeof(struct sockaddr) ) ;

	b = GetHttpRequestBuffer(e) ;
	nret = StrcatHttpBuffer( b , str ) ;

	nret = RequestHttp( connect_sock , NULL , e ) ;
	CLOSESOCKET( connect_sock );

	return 0;
}

int test_client_block()
{
	struct HttpEnv		*e = NULL ;

	int			nret = 0 ;

	e = CreateHttpEnv();

	nret = TestParseHttpRequest( e , "GET / HTTP/1.1\r\n"
					"Host: www.baidu.com\r\n"
					"User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:45.0) Gecko/20100101
Firefox/45.0\r\n"
					"Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3\r\n"
					"Accept-Encoding: gzip, deflate, br\r\n"
					"Cookie: BAIDUID=0E27B789D33BF3C43C6022BD0182CF8D:SL=0:NR=10:FG=1;
BIDUPSID=EE65333C3C1B7FB4807F6DC5DE576979; PSTM=1462883721; BD_UPN=13314152; ispeed_lsm=2;
MCITY=-179%3A;
BDUSS=t4TW1VRFNsMm91bGtTcUFHbVFqfnhiVFVYd2ZKZFc2c0dGaG12VmhZckZJbmxYQVFBQUFBJCQAAAAAAAA
```

```
AAAEAAADIZsc0Y2FsdmlubGljaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAMWVUVfFlVFXSG; pgv_pvi=56303616; BD_HOME=1;
H_PS_PSSID=19290_1436_18240_20076_17001_15790_12201_20254; sug=3; sugstore=0; ORIGIN=2; bdime=0;
__bsi=13900513390515515511_00_0_I_R_33_0303_C02F_N_I_I_0\r\n"
					"Connection: keep-alive\r\n"
					"Cache-Control: max-age=0\r\n"
					"\r\n" ) ;

	DestroyHttpEnv( e );

	printf( "ALL test is ok!!!\n" );

	return 0;
}

int main()
{
#if ( defined _WIN32 )
	WSADATA		wsaData;
#endif
	int		nret = 0 ;

#if ( defined _WIN32 )
	nret = WSAStartup( MAKEWORD( 2, 2 ), &wsaData ) ;
	if( nret )
	{
		printf( "WSAStartup failed[%d] , errno[%d]\n" , nret , GetLastError() );
		return 1;
	}
#endif

	nret = test_client_block() ;

#if ( defined _WIN32 )
	WSACleanup();
#endif

	return -nret;
}
```

服务端（fasterhttp API 加粗显示）（位于源代码包 test/test_server_block.c）

```
#include "fasterhttp.h"

funcProcessHttpRequest ProcessHttpRequest ;
int ProcessHttpRequest( struct HttpEnv *e , void *p )
```

```
{
	struct HttpBuffer *b = NULL ;
	int			nret = 0 ;

	b = GetHttpResponseBuffer(e) ;
	nret = StrcatHttpBuffer( b ,  "Content-Type: text/html\r\n"
				"Content-Length: 17\r\n"
				"\r\n"
				"hello fasterhttp!" ) ;

	return HTTP_OK;
}

int test_server_block()
{
	SOCKET			listen_sock ;
	struct sockaddr_in	listen_addr ;
	SOCKET			accept_sock ;
	struct sockaddr_in	accept_addr ;
	SOCKLEN_T		accept_addr_len ;
	int			onoff ;

	struct HttpEnv		*e = NULL ;

	int			nret = 0 ;

	listen_sock = socket( AF_INET , SOCK_STREAM , IPPROTO_TCP ) ;

	onoff = 1 ;
	setsockopt( listen_sock , SOL_SOCKET , SO_REUSEADDR , (void *) & onoff , sizeof(onoff) );

	memset( & listen_addr , 0x00 , sizeof(struct sockaddr_in) );
	listen_addr.sin_family = AF_INET;
	listen_addr.sin_addr.s_addr = INADDR_ANY ;
	listen_addr.sin_port = htons( (unsigned short)9527 );

	nret = bind( listen_sock , (struct sockaddr *) & listen_addr , sizeof(struct sockaddr) ) ;

	nret = listen( listen_sock , 1024 ) ;

	while(1)
	{
		accept_addr_len = sizeof(struct sockaddr) ;
		accept_sock = accept( listen_sock , (struct sockaddr *) & accept_addr, & accept_addr_len );
```

```
		e = CreateHttpEnv();

		EnableHttpResponseCompressing( e , 1 );

		nret = ResponseAllHttp( accept_sock , NULL , e , & ProcessHttpRequest , (void*)(&accept_sock) ) ;

		DestroyHttpEnv( e );
		CLOSESOCKET( accept_sock );
	}

	CLOSESOCKET( listen_sock );

	return 0;
}

int main()
{
#if ( defined _WIN32 )
	WSADATA		wsaData;
#endif
	int		nret = 0 ;

	setbuf( stdout , NULL );

#if ( defined _WIN32 )
	nret = WSAStartup( MAKEWORD( 2, 2 ), &wsaData ) ;
	if( nret )
	{
		printf( "WSAStartup failed[%d] , errno[%d]\n" , nret , GetLastError() );
		return 1;
	}
#endif

	nret = test_server_block() ;

#if ( defined _WIN32 )
	WSACleanup();
#endif

	return -nret;
}
```

3.2 非堵塞/多路复用模型

基本使用流程和堵塞模型差不多。

客户端的 RequestHttp 拆分成 SendHttpRequestNonblock 和 ReceiveHttpResponseNonblock 非堵塞调用。

服务端的 ResponseHttp 拆分成 ReceiveHttpRequestNonblock 和 SendHttpResponseNonblock 非堵塞调用。HTTP 请求处理函数 ProcessHttpRequest 返回错误时，调用 FormatHttpResponseStartLine 自行组织报错响应报文。

下面是非堵塞/多路复用（基于 select）的客户端和服务端示例（为突出显示主干，删除了所有出错处理；fasterhttp API 加粗显示），客户端先（位于源代码包 test/test_client_nonblock.c）：

```
#include "fasterhttp.h"

static int TestParseHttpRequest( struct HttpEnv *e , char *str )
{
	SOCKET			connect_sock ;
	struct sockaddr_in	connect_addr ;

	struct HttpBuffer *b = NULL ;

	int			nret = 0 ;

	ResetHttpEnv( e );

	connect_sock = socket( AF_INET , SOCK_STREAM , IPPROTO_TCP ) ;

	memset( & connect_addr , 0x00 , sizeof(struct sockaddr_in) );
	connect_addr.sin_family = AF_INET;
	connect_addr.sin_addr.s_addr = inet_addr( "127.0.0.1" );
	connect_addr.sin_port = htons( (unsigned short)9527 );

	nret = connect( connect_sock , (struct sockaddr *) & connect_addr , sizeof(struct sockaddr) ) ;

	SetHttpNonblock( connect_sock );

	b = GetHttpRequestBuffer(e) ;
	nret = StrcatHttpBuffer( b , str ) ;

	while(1)
```

```
	{
		nret = SendHttpRequestNonblock( connect_sock , NULL , e ) ;
		if( nret == FASTERHTTP_INFO_TCP_SEND_WOULDBLOCK )
		{
			;
		}
		else if( nret )
		{
			printf( "SendHttpRequestNonblock failed[%d]\n" , nret );
			CLOSESOCKET( connect_sock );
			return nret;
		}
		else
		{
			break;
		}
	}

	while(1)
	{
		nret = ReceiveHttpResponseNonblock( connect_sock , NULL , e ) ;
		if( nret == FASTERHTTP_INFO_NEED_MORE_HTTP_BUFFER )
		{
			;
		}
		else if( nret )
		{
			printf( "ReceiveHttpResponseNonblock failed[%d]\n" , nret );
			CLOSESOCKET( connect_sock );
			return nret;
		}
		else
		{
			break;
		}
	}

	CLOSESOCKET( connect_sock );

	return 0;
}

int test_client_nonblock()
{
```

```
	struct HttpEnv		*e = NULL ;

	int			nret = 0 ;

	e = CreateHttpEnv();

	nret = TestParseHttpRequest( e , "GET / HTTP/1.1\r\n"
					"Host: www.baidu.com\r\n"
					"User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:45.0) Gecko/20100101
Firefox/45.0\r\n"
					"Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3\r\n"
					"Accept-Encoding: gzip, deflate, br\r\n"
					"Cookie: BAIDUID=0E27B789D33BF3C43C6022BD0182CF8D:SL=0:NR=10:FG=1;
BIDUPSID=EE65333C3C1B7FB4807F6DC5DE576979; PSTM=1462883721; BD_UPN=13314152; ispeed_lsm=2;
MCITY=-179%3A;
BDUSS=t4TW1VRFNsMm91bGtTcUFHbVFqfnhiVFVYd2ZKZFc2c0dGaG12VmhZckZJbmxYQVFBQUFBJCQAAAAAAAA
AAAEAAADIZsc0Y2FsdmlubGljaAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAMWVUVfFlVFXSG; pgv_pvi=56303616; BD_HOME=1;
H_PS_PSSID=19290_1436_18240_20076_17001_15790_12201_20254; sug=3; sugstore=0; ORIGIN=2; bdime=0;
__bsi=13900513390515515511_00_0_I_R_33_0303_C02F_N_I_I_0\r\n"
					"Connection: keep-alive\r\n"
					"Cache-Control: max-age=0\r\n"
					"\r\n" ) ;

	DestroyHttpEnv( e );

	printf( "ALL test is ok!!!\n" );

	return 0;
}

int main()
{
#if ( defined _WIN32 )
	WSADATA		wsaData;
#endif
	int		nret = 0 ;

#if ( defined _WIN32 )
	nret = WSAStartup( MAKEWORD( 2, 2 ), &wsaData ) ;
	if( nret )
	{
		printf( "WSAStartup failed[%d] , errno[%d]\n" , nret , GetLastError() );
		return 1;
```

```
	}
#endif

	nret = test_client_nonblock() ;

#if ( defined _WIN32 )
	WSACleanup();
#endif

	return -nret;
}
```

服务端（fasterhttp API 加粗显示）（位于源代码包 test/test_server_nonblock.c）

```
#include "fasterhttp.h"

funcProcessHttpRequest ProcessHttpRequest ;
int ProcessHttpRequest( struct HttpEnv *e , void *p )
{
	struct HttpBuffer *b = NULL ;
	int		nret = 0 ;

	b = GetHttpResponseBuffer(e) ;
	nret = StrcatHttpBuffer( b ,  "Content-Type: text/html\r\n"
				"Content-Length: 17\r\n"
				"\r\n"
				"hello fasterhttp!" ) ;

	return HTTP_OK;
}

int test_server_nonblock()
{
	SOCKET			listen_sock ;
	struct sockaddr_in	listen_addr ;
	SOCKET			accept_sock ;
	struct sockaddr_in	accept_addr ;
	SOCKLEN_T		accept_addr_len ;
	int		onoff ;

	struct HttpEnv		*e = NULL ;

	int		nret = 0 ;

	listen_sock = socket( AF_INET , SOCK_STREAM , IPPROTO_TCP ) ;
```

```
onoff = 1 ;
setsockopt( listen_sock , SOL_SOCKET , SO_REUSEADDR , (void *) & onoff , sizeof(onoff) );

memset( & listen_addr , 0x00 , sizeof(struct sockaddr_in) );
listen_addr.sin_family = AF_INET;
listen_addr.sin_addr.s_addr = INADDR_ANY ;
listen_addr.sin_port = htons( (unsigned short)9527 );

nret = bind( listen_sock , (struct sockaddr *) & listen_addr , sizeof(struct sockaddr) ) ;

nret = listen( listen_sock , 1024 ) ;

while(1)
{
	accept_addr_len = sizeof(struct sockaddr) ;
	accept_sock = accept( listen_sock , (struct sockaddr *) & accept_addr, & accept_addr_len );

	SetHttpNonblock( accept_sock );

	e = CreateHttpEnv();

	EnableHttpResponseCompressing( e , 1 );

	while(1)
	{
		while(1)
		{
			nret = ReceiveHttpRequestNonblock( accept_sock , NULL , e ) ;
			if( nret == FASTERHTTP_INFO_NEED_MORE_HTTP_BUFFER )
			{
				;
			}
			else
			{
				break;
			}
		}

		if( nret == FASTERHTTP_ERROR_TCP_CLOSE )
		{
			break;
		}
		else if( nret == FASTERHTTP_INFO_TCP_CLOSE )
		{
```

```
		break;
	}
	else if( nret )
	{
		printf( "ReceiveHttpRequestNonblock failed[%d]\n" , nret );

		nret = FormatHttpResponseStartLine( abs(nret)/1000 , e , 1 ) ;
		if( nret )
			break;
	}
	else
	{
		nret = FormatHttpResponseStartLine( HTTP_OK , e , 0 ) ;
		if( nret )
			break;

		nret = ProcessHttpRequest( e , (void*)(&accept_sock) ) ;
		if( nret != HTTP_OK )
		{
			nret = FormatHttpResponseStartLine( nret , e , 1 ) ;
			if( nret )
				break;
		}
	}

	while(1)
	{
		nret = SendHttpResponseNonblock( accept_sock , NULL , e ) ;
		if( nret == FASTERHTTP_INFO_TCP_SEND_WOULDBLOCK )
		{
			;
		}
		else
		{
			break;
		}
	}

	if( nret )
	{
		printf( "SendHttpResponseNonblock failed[%d]\n" , nret );
		break;
	}
```

```
                    if( ! CheckHttpKeepAlive(e) )
                        break;

                    ResetHttpEnv(e);
                }

                DestroyHttpEnv( e );
                CLOSESOCKET( accept_sock );
        }

        CLOSESOCKET( listen_sock );

        return 0;
}

int main()
{
#if ( defined _WIN32 )
        WSADATA         wsaData;
#endif
        int             nret = 0 ;

#if ( defined _WIN32 )
        nret = WSAStartup( MAKEWORD( 2, 2 ), &wsaData ) ;
        if( nret )
        {
                printf( "WSAStartup failed[%d] , errno[%d]\n" , nret , GetLastError() );
                return 1;
        }
#endif

        nret = test_server_nonblock() ;

#if ( defined _WIN32 )
        WSACleanup();
#endif

        return -nret;
}
```

3.3 一个基于 epoll 的小型 HTTP 服务器

下面是一个基于 epoll 多路复用的服务端示例（源代码位于 demo/htmlserver/htmlserver.c）

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/epoll.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>

#include "fasterhttp.h"
#include "LOGC.h"

#define MAX_EPOLL_EVENTS        1024

static int ProcessHttpRequest( struct HttpEnv *e , int sock , char *wwwroot )
{
        char                    pathfilename[ 1024 + 1 ] ;
        struct stat             st ;
        int                     filesize ;
        struct HttpBuffer *b = NULL ;

        SOCKLEN_T               socklen ;
        struct sockaddr_in      client_sockaddr ;
        char                    client_ip[ 15 + 1 ] ;
        int                     client_port ;
        struct sockaddr_in      server_sockaddr ;
        char                    server_ip[ 15 + 1 ] ;
        int                     server_port ;

        int                     nret = 0 ;

        memset( pathfilename , 0x00 , sizeof(pathfilename) );
        snprintf( pathfilename , sizeof(pathfilename)-1 , "%s%.*s" , wwwroot , GetHttpHeaderLen_URI(e) ,
GetHttpHeaderPtr_URI(e,NULL) );
```

```
nret = stat( pathfilename , & st ) ;
if( nret == -1 )
	return HTTP_NOT_FOUND;
filesize = st.st_size ;

b = GetHttpResponseBuffer(e) ;
nret = StrcatfHttpBuffer( b ,  "Server: htmlserver/1.0.0\r\n"
				"Content-Type: text/html\r\n"
				"Content-Length: %d\r\n"
				"\r\n"
				, filesize ) ;
if( nret )
	return HTTP_INTERNAL_SERVER_ERROR;

nret = StrcatHttpBufferFromFile( b , pathfilename , &filesize ) ;
if( nret )
	return HTTP_INTERNAL_SERVER_ERROR;

socklen = sizeof(struct sockaddr) ;
nret = getpeername( sock , (struct sockaddr *) & client_sockaddr , & socklen ) ;
if( nret )
{
	printf( "getpeername failed , errno[%d]\n" , errno );
	return HTTP_INTERNAL_SERVER_ERROR;
}
memset( client_ip , 0x00 , sizeof(client_ip) );
inet_ntop( AF_INET , &(client_sockaddr.sin_addr) , client_ip , sizeof(client_ip) );
client_port = (int)ntohs(client_sockaddr.sin_port) ;

socklen = sizeof(struct sockaddr) ;
nret = getsockname( sock , (struct sockaddr *) & server_sockaddr , & socklen ) ;
if( nret )
{
	printf( "getsockname failed , errno[%d]\n" , errno );
	return HTTP_INTERNAL_SERVER_ERROR;
}
memset( server_ip , 0x00 , sizeof(server_ip) );
inet_ntop( AF_INET , &(server_sockaddr.sin_addr) , server_ip , sizeof(server_ip) );
server_port = (int)ntohs(server_sockaddr.sin_port) ;

InfoLog( __FILE__ , __LINE__ , "%s:%d -> %s:%d | %.*s %.*s %.*s 200"
	, client_ip , client_port , server_ip , server_port
	, GetHttpHeaderLen_METHOD(e) , GetHttpHeaderPtr_METHOD(e,NULL)
```

```
		, GetHttpHeaderLen_URI(e) , GetHttpHeaderPtr_URI(e,NULL)
		, GetHttpHeaderLen_VERSION(e) , GetHttpHeaderPtr_VERSION(e,NULL)
		);

	return HTTP_OK;
}

static int OnAcceptingSocket( int epoll_fd , int listen_sock )
{
	SOCKET			accept_sock ;
	struct sockaddr_in	accept_addr ;
	SOCKLEN_T		accept_addr_len ;

	struct epoll_event	event ;

	struct HttpEnv		*e = NULL ;

	int			nret = 0 ;

	accept_addr_len = sizeof(struct sockaddr) ;
	accept_sock = accept( listen_sock , (struct sockaddr *) & accept_addr, & accept_addr_len );
	if( accept_sock == - 1 )
	{
		ErrorLog( __FILE__ , __LINE__ , "accept failed , errno[%d]" , errno );
		return -1;
	}
	else
	{
		DebugLog( __FILE__ , __LINE__ , "accept ok" );
	}

	SetHttpNonblock( accept_sock );

	e = CreateHttpEnv() ;
	if( e == NULL )
	{
		ErrorLog( __FILE__ , __LINE__ , "CreateHttpEnv failed , errno[%d]" , errno );
		return -1;
	}

	SetHttpTimeout( e , 120 );
	EnableHttpResponseCompressing( e , 1 );

	memset( & event , 0x00 , sizeof(struct epoll_event) );
```

```
	event.events = EPOLLIN | EPOLLERR ;
	event.data.ptr = e ;
	SetParserCustomIntData( e , accept_sock );
	nret = epoll_ctl( epoll_fd , EPOLL_CTL_ADD , accept_sock , & event ) ;
	if( nret == -1 )
	{
		ErrorLog( __FILE__ , __LINE__ , "epoll_ctl failed , errno[%d]" , errno );
		return -1;
	}

	return 0;
}

static int OnReceivingSocket( int epoll_fd , int accept_sock , struct HttpEnv *e , char *wwwroot )
{
	struct epoll_event	event ;

	int			nret = 0 ;

	nret = ReceiveHttpRequestNonblock( accept_sock , NULL , e ) ;
	if( nret == FASTERHTTP_INFO_NEED_MORE_HTTP_BUFFER )
	{
		;
	}
	else if( nret )
	{
		if( nret == FASTERHTTP_ERROR_TCP_CLOSE )
		{
			ErrorLog( __FILE__ , __LINE__ , "accepted socket closed detected" );
			return -1;
		}
		else if( nret == FASTERHTTP_INFO_TCP_CLOSE )
		{
			InfoLog( __FILE__ , __LINE__ , "accepted socket closed detected" );
			return -1;
		}
		else
		{
			ErrorLog( __FILE__ , __LINE__ , "ReceiveHttpRequestNonblock failed[%d] , errno[%d]" , nret ,
errno );

			nret = FormatHttpResponseStartLine( abs(nret)/1000 , e , 1 ) ;
			if( nret )
			{
```

```
                        ErrorLog( __FILE__ , __LINE__ , "FormatHttpResponseStartLine failed[%d] , errno[%d]" ,
nret , errno );
                        return -2;
                    }

                    return 0;
                }
            }
            else
            {
                nret = FormatHttpResponseStartLine( HTTP_OK , e , 0 ) ;
                if( nret )
                {
                    ErrorLog( __FILE__ , __LINE__ , "FormatHttpResponseStartLine failed[%d] , errno[%d]" , nret ,
errno );
                    return -2;
                }

                nret = ProcessHttpRequest( e , GetParserCustomIntData(e) , wwwroot ) ;
                if( nret != HTTP_OK )
                {
                    nret = FormatHttpResponseStartLine( nret , e , 1 ) ;
                    if( nret )
                    {
                        ErrorLog( __FILE__ , __LINE__ , "FormatHttpResponseStartLine failed[%d] , errno[%d]" ,
nret , errno );
                        return -2;
                    }
                }

                memset( & event , 0x00 , sizeof(struct epoll_event) );
                event.events = EPOLLOUT | EPOLLERR ;
                event.data.ptr = e ;
                nret = epoll_ctl( epoll_fd , EPOLL_CTL_MOD , accept_sock , & event ) ;
                if( nret == -1 )
                {
                    ErrorLog( __FILE__ , __LINE__ , "epoll_ctl failed , errno[%d]" , errno );
                    return -2;
                }
            }

            return 0;
}
```

```
static int OnSendingSocket( int epoll_fd , int accept_sock , struct HttpEnv *e )
{
	struct epoll_event	event ;

	int			nret = 0 ;

	nret = SendHttpResponseNonblock( accept_sock , NULL , e ) ;
	if( nret == FASTERHTTP_INFO_TCP_SEND_WOULDBLOCK )
	{
		;
	}
	else if( nret )
	{
		ErrorLog( __FILE__ , __LINE__ , "SendHttpResponseNonblock failed[%d] , errno[%d]" , nret , errno );
		return -1;
	}
	else
	{
		if( CheckHttpKeepAlive(e) )
		{
			ResetHttpEnv(e);

			memset( & event , 0x00 , sizeof(struct epoll_event) );
			event.events = EPOLLIN | EPOLLERR ;
			event.data.ptr = e ;
			nret = epoll_ctl( epoll_fd , EPOLL_CTL_MOD , accept_sock , & event ) ;
			if( nret == -1 )
			{
				ErrorLog( __FILE__ , __LINE__ , "epoll_ctl failed , errno[%d]" , errno );
				return -2;
			}
		}
		else
		{
			DebugLog( __FILE__ , __LINE__ , "close client socket" );
			return -1;
		}
	}

	return 0;
}

static int htmlserver( int port , char *wwwroot )
{
```

```
int                     epoll_fd ;
struct epoll_event      event , *p_event = NULL ;
struct epoll_event      events[ MAX_EPOLL_EVENTS ] ;
int                     nfds , i ;

struct HttpEnv          *e = NULL ;

SOCKET                  listen_sock ;
struct sockaddr_in      listen_addr ;
int                     onoff ;

int                     nret = 0 ;

epoll_fd = epoll_create( 1024 ) ;
if( epoll_fd == -1 )
{
        ErrorLog( __FILE__ , __LINE__ , "epoll_create failed , errno[%d]" , errno );
        return -1;
}
else
{
        InfoLog( __FILE__ , __LINE__ , "epoll_create ok" );
}

listen_sock = socket( AF_INET , SOCK_STREAM , IPPROTO_TCP ) ;
if( listen_sock == -1 )
{
        ErrorLog( __FILE__ , __LINE__ , "socket failed , errno[%d]" , errno );
        CLOSESOCKET( epoll_fd );
        return -1;
}
else
{
        InfoLog( __FILE__ , __LINE__ , "socket ok" );
}

onoff = 1 ;
setsockopt( listen_sock , SOL_SOCKET , SO_REUSEADDR , (void *) & onoff , sizeof(onoff) );

memset( & listen_addr , 0x00 , sizeof(struct sockaddr_in) ) ;
listen_addr.sin_family = AF_INET ;
listen_addr.sin_addr.s_addr = INADDR_ANY ;
listen_addr.sin_port = htons( (unsigned short)port ) ;
```

```
nret = bind( listen_sock , (struct sockaddr *) & listen_addr , sizeof(struct sockaddr) ) ;
if( nret == -1 )
{
	ErrorLog( __FILE__ , __LINE__ , "bind failed , errno[%d]" , errno );
	CLOSESOCKET( listen_sock );
	CLOSESOCKET( epoll_fd );
	return -1;
}
else
{
	InfoLog( __FILE__ , __LINE__ , "bind ok" );
}

nret = listen( listen_sock , 1024 ) ;
if( nret == -1 )
{
	ErrorLog( __FILE__ , __LINE__ , "listen failed , errno[%d]" , errno );
	CLOSESOCKET( listen_sock );
	CLOSESOCKET( epoll_fd );
	return -1;
}
else
{
	InfoLog( __FILE__ , __LINE__ , "listen ok" );
}

memset( & event , 0x00 , sizeof(struct epoll_event) );
event.events = EPOLLIN | EPOLLERR ;
event.data.ptr = NULL ;
nret = epoll_ctl( epoll_fd , EPOLL_CTL_ADD , listen_sock , & event ) ;
if( nret == -1 )
{
	ErrorLog( __FILE__ , __LINE__ , "epoll_ctl failed , errno[%d]" , errno );
	CLOSESOCKET( listen_sock );
	CLOSESOCKET( epoll_fd );
	return -1;
}
else
{
	InfoLog( __FILE__ , __LINE__ , "epoll_ctl ok" );
}

while(1)
{
```

```
		memset( events , 0x00 , sizeof(events) );
		nfds = epoll_wait( epoll_fd , events , MAX_EPOLL_EVENTS , -1 ) ;
		if( nfds == -1 )
		{
			ErrorLog( __FILE__ , __LINE__ , "epoll_wait failed , errno[%d]" , errno );
			CLOSESOCKET( listen_sock );
			CLOSESOCKET( epoll_fd );
			return -1;
		}

		for( i = 0 , p_event = events ; i < nfds ; i++ , p_event++ )
		{
			if( p_event->data.ptr == NULL )
			{
				if( p_event->events & EPOLLIN )
				{
					nret = OnAcceptingSocket( epoll_fd , listen_sock ) ;
					if( nret == -1 )
					{
						ErrorLog( __FILE__ , __LINE__ , "OnAcceptingSocket failed , errno[%d]" ,
errno );
						CLOSESOCKET( listen_sock );
						CLOSESOCKET( epoll_fd );
						return nret;
					}
				}
				else if( p_event->events & EPOLLERR )
				{
					ErrorLog( __FILE__ , __LINE__ , "listen_sock epoll EPOLLERR" );
					CLOSESOCKET( listen_sock );
					CLOSESOCKET( epoll_fd );
					return -1;
				}
				else
				{
					ErrorLog( __FILE__ , __LINE__ , "listen_sock epoll event invalid[%d]" ,
p_event->events );
					CLOSESOCKET( listen_sock );
					CLOSESOCKET( epoll_fd );
					return -1;
				}
			}
			else
			{
```

```
int     accept_sock ;

e = p_event->data.ptr ;
accept_sock = GetParserCustomIntData(e) ;

if( p_event->events & EPOLLIN )
{
        nret = OnReceivingSocket( epoll_fd , accept_sock , e , wwwroot ) ;
        if( nret == -1 )
        {
                DestroyHttpEnv(e);
                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                CLOSESOCKET( accept_sock );
        }
        else if( nret == -2 )
        {
                DestroyHttpEnv(e);
                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                CLOSESOCKET( accept_sock );
                CLOSESOCKET( listen_sock );
                CLOSESOCKET( epoll_fd );
                return nret;
        }
}
else if( p_event->events & EPOLLOUT )
{
        nret = OnSendingSocket( epoll_fd , accept_sock , e ) ;
        if( nret == -1 )
        {
                DestroyHttpEnv(e);
                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                CLOSESOCKET( accept_sock );
        }
        else if( nret == -2 )
        {
                DestroyHttpEnv(e);
                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                CLOSESOCKET( accept_sock );
                CLOSESOCKET( listen_sock );
                CLOSESOCKET( epoll_fd );
                return nret;
        }
}
else if( p_event->events & EPOLLERR )
```

```
                        {
                                ErrorLog( __FILE__ , __LINE__ , "accept_sock epoll EPOLLERR" );
                                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                                CLOSESOCKET( accept_sock );
                        }
                        else
                        {
                                ErrorLog( __FILE__ , __LINE__ , "accept_sock epoll event invalid[%d]" ,
p_event->events );
                                epoll_ctl( epoll_fd , EPOLL_CTL_DEL , accept_sock , NULL );
                                CLOSESOCKET( accept_sock );
                        }
                }
        }
    }

    CLOSESOCKET( listen_sock );
    CLOSESOCKET( epoll_fd );

    return 0;
}

static void usage()
{
    printf( "USAGE : htmlserver port\n" );
    return;
}

int main( int argc , char *argv[] )
{
    SetLogFile( "%s/log/htmlserver.log" , getenv("HOME") );
    SetLogLevel( LOGLEVEL_INFO );

    ResetAllHttpStatus();
    SetHttpStatus( HTTP_NOT_FOUND , HTTP_NOT_FOUND_S , "Custem Not Found Text" );

    if( argc == 1 + 2 )
    {
        return -htmlserver( atoi(argv[1]) , argv[2] );
    }
    else
    {
        usage();
        exit(9);
```

```
    }
}
```

4 性能压测

4.1 压测方案

压测环境

CPU : Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz

内存 : 1GB

硬盘 : （足够）

带宽 : 1MB

操作系统 : CentOS release 6.5 (Final) 64bits

同类软件

http-parser-2.7.0	使用最广泛的 HTTP 解析器
fasterhttp-1.0.0	本文的主角
picohttpparser(2014)	声称性能最快的 HTTP 解析器

压测过程

压测比较三个 HTTP 解析库

只压测 HTTP 请求数据的解析，使用各自自带的压测程序，HTTP 请求数据修改成一样

三个 HTTP 解析库交叉运行 11 轮，每轮解析 1000 万次 HTTP 请求，取 Linux 的 time 数据为准

4.2 压测结果

http-parser 压测结果如下：

```
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.753361 seconds to run
```

```
533237.750000 req/sec

real    0m18.755s
user    0m18.700s
sys     0m0.037s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.832979 seconds to run
530983.437500 req/sec

real    0m18.834s
user    0m18.786s
sys     0m0.030s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.816858 seconds to run
531438.312500 req/sec

real    0m18.818s
user    0m18.748s
sys     0m0.045s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.853455 seconds to run
530406.750000 req/sec

real    0m18.855s
user    0m18.787s
sys     0m0.050s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.871876 seconds to run
529889.062500 req/sec

real    0m18.873s
user    0m18.812s
sys     0m0.035s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.853525 seconds to run
530404.812500 req/sec

real    0m18.855s
user    0m18.777s
```

```
sys         0m0.060s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.877413 seconds to run
529733.625000 req/sec

real        0m18.879s
user        0m18.816s
sys         0m0.044s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.975199 seconds to run
527003.687500 req/sec

real        0m18.976s
user        0m18.915s
sys         0m0.044s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.987669 seconds to run
526657.562500 req/sec

real        0m18.989s
user        0m18.918s
sys         0m0.044s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.821001 seconds to run
531321.375000 req/sec

real        0m18.822s
user        0m18.753s
sys         0m0.051s
[calvin@iZ23k0yd363Z /home/calvin/expack/http-parser] time ./bench2 10000000
Benchmark result:
Took 18.804218 seconds to run
531795.562500 req/sec

real        0m18.805s
user        0m18.751s
sys         0m0.037s
```

fasterhttp 压测结果如下：

```
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.041s
user    0m11.005s
sys     0m0.026s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.009s
user    0m10.962s
sys     0m0.029s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.037s
user    0m11.000s
sys     0m0.026s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.140s
user    0m11.101s
sys     0m0.028s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.041s
user    0m11.005s
sys     0m0.025s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.779s
user    0m11.741s
sys     0m0.028s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.027s
user    0m10.989s
sys     0m0.026s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.203s
user    0m11.158s
sys     0m0.033s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real    0m11.043s
user    0m10.999s
```

```
sys         0m0.033s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real        0m11.054s
user        0m11.013s
sys         0m0.030s
[calvin@iZ23k0yd363Z /home/calvin/exsrc/fasterhttp/test] time ./press 10000000

real        0m11.035s
user        0m10.992s
sys         0m0.033s
```

picohttpparser 压测结果如下：

```
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m8.001s
user        0m7.970s
sys         0m0.016s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m7.918s
user        0m7.880s
sys         0m0.024s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m7.918s
user        0m7.898s
sys         0m0.013s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m7.964s
user        0m7.941s
sys         0m0.015s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m8.035s
user        0m8.012s
sys         0m0.017s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real        0m7.972s
user        0m7.947s
sys         0m0.017s
```

```
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real    0m7.980s
user    0m7.956s
sys     0m0.017s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real    0m7.980s
user    0m7.959s
sys     0m0.014s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real    0m7.991s
user    0m7.963s
sys     0m0.021s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real    0m7.960s
user    0m7.934s
sys     0m0.020s
[calvin@iZ23k0yd363Z /home/calvin/expack/picohttpparser] time ./bench

real    0m7.975s
user    0m7.933s
sys     0m0.027s
```

4.3 压测评价

原来不想把 picohttpparser 纳入同类比较，因为 picohttpparser 存在设计缺陷，原因如下：

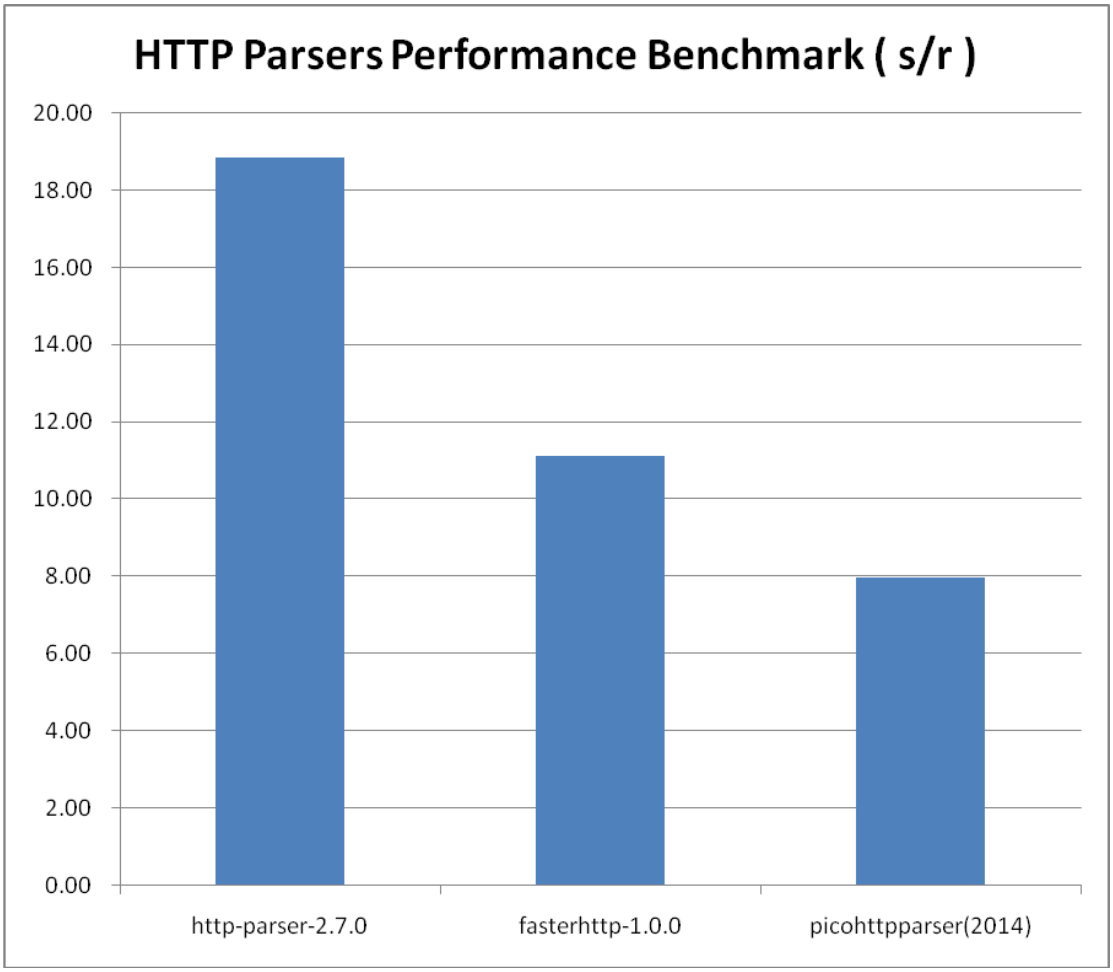
1.单纯的只进行断句，不判断 HTTP 请求是否完整，不识别任何 HTTP 头选项，不对 HTTP 头选项进行合法性检查和语义处理，甚至包括 Content-Length，使用者会问："难道要我自己解析 Content-Length 以便知道 HTTP 请求是否完整了？那可是一个 HTTP 解析器的核心功能啊，我自己都解析了还要 HTTP 解析器干什么？"，没错，picohttpparser 就是这样设计的，这就形成了一个悖论，谁使用谁倒霉。

2.不支持非堵塞流式解析，当网络质量不好时，就得一遍又一遍的重复解析

增量后的全量数据，性能反而很差 。

但我最后还是把它放进来是为了看看 fasterhttp 离它的性能差距到底有多大。

下图就是压测跑出来的结果（平均值）



可以看出，功能性差不多的 fasterhttp 在性能上比 http-parser 快了近一倍，逼近功能不完整的 picohttpparser 的性能了（做的事少当然速度就快了，呵呵）。

恩，总体满意，以后有想法再继续优化吧。

1.1 压测代码

http-parser 自带的 bench.c 修改统一了 HTTP 请求报文

```
/* Copyright Fedor Indutny. All rights reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
 * of this software and associated documentation files (the "Software"), to
 * deal in the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */
#include "http_parser.h"
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>

static const char data[] =
        "GET /wp-content/uploads/2010/03/hello-kitty-darth-vader-pink.jpg HTTP/1.1\r\n"
\
        "Host: www.kittyhell.com\r\n"
\
        "User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; ja-JP-mac; rv:1.9.2.3) Gecko/20100401
Firefox/3.6.3 "                         \
        "Pathtraq/0.9\r\n"
\
        "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
\
        "Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n"
\
        "Accept-Encoding: gzip,deflate\r\n"
\
        "Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7\r\n"
\
        "Keep-Alive: 115\r\n"
\
        "Connection: keep-alive\r\n"
```

```
\
       "Cookie: wp_ozh_wsa_visits=2; wp_ozh_wsa_visit_lasttime=xxxxxxxxxx; "
\
       "__utma=xxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.x; "
\

"__utmz=xxxxxxxxx.xxxxxxxxxx.x.x.utmccn=(referral)|utmcsr=reader.livedoor.com|utmcct=/reader/|utmcmd=ref
erral\r\n"                          \
       "\r\n" ;
static const size_t data_len = sizeof(data) - 1;

static int on_info(http_parser* p) {
   return 0;
}



static int on_data(http_parser* p, const char *at, size_t length) {
   return 0;
}

static http_parser_settings settings = {
   .on_message_begin = on_info,
   .on_headers_complete = on_info,
   .on_message_complete = on_info,
   .on_header_field = on_data,
   .on_header_value = on_data,
   .on_url = on_data,
   .on_status = on_data,
   .on_body = on_data
};

int bench(int iter_count, int silent) {
   struct http_parser parser;
   int i;
   int err;
   struct timeval start;
   struct timeval end;
   float rps;

   if (!silent) {
      err = gettimeofday(&start, NULL);
      assert(err == 0);
   }
```

```
  for (i = 0; i < iter_count; i++) {
    size_t parsed;
    http_parser_init(&parser, HTTP_REQUEST);

    parsed = http_parser_execute(&parser, &settings, data, data_len);
    assert(parsed == data_len);
  }

  if (!silent) {
    err = gettimeofday(&end, NULL);
    assert(err == 0);

    fprintf(stdout, "Benchmark result:\n");

    rps = (float) (end.tv_sec - start.tv_sec) +
          (end.tv_usec - start.tv_usec) * 1e-6f;
    fprintf(stdout, "Took %f seconds to run\n", rps);

    rps = (float) iter_count / rps;
    fprintf(stdout, "%f req/sec\n", rps);
    fflush(stdout);
  }

  return 0;
}

int main(int argc, char** argv) {
  if (argc == 2 && strcmp(argv[1], "infinite") == 0) {
    for (;;)
      bench(5000000, 1);
    return 0;
  } else {
    return bench(atoi(argv[1]), 0);
  }
}
```

fasterhttp 自带的压测代码，位于 test/press.c

```
#include "fasterhttp.h"

#define REQ
\
    "GET /wp-content/uploads/2010/03/hello-kitty-darth-vader-pink.jpg HTTP/1.1\r\n"
\
```

```
	"Host: www.kittyhell.com\r\n"
\
	"User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; ja-JP-mac; rv:1.9.2.3) Gecko/20100401
Firefox/3.6.3 "			\
	"Pathtraq/0.9\r\n"
\
	"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
\
	"Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n"
\
	"Accept-Encoding: gzip,deflate\r\n"
\
	"Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7\r\n"
\
	"Keep-Alive: 115\r\n"
\
	"Connection: keep-alive\r\n"
\
	"Cookie: wp_ozh_wsa_visits=2; wp_ozh_wsa_visit_lasttime=xxxxxxxxxx; "
\
	"__utma=xxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.x; "
\

"__utmz=xxxxxxxxx.xxxxxxxxxx.x.x.utmccn=(referral)|utmcsr=reader.livedoor.com|utmcct=/reader/|utmcmd=ref
erral\r\n"			\
	"\r\n"

static int press( int count )
{
	struct HttpEnv		*e = NULL ;
	int			i ;
	struct HttpBuffer	*b = NULL ;
	int			nret = 0 ;

	e = CreateHttpEnv() ;
	if( e == NULL )
	{
		printf( "CreateHttpEnv failed\n" );
		return -1;
	}

	for( i = 0 ; i < count ; i++ )
	{
		ResetHttpEnv( e );
```

```
                b = GetHttpRequestBuffer( e ) ;
                SetHttpBufferPtr( b , REQ , sizeof(REQ) ) ;

                nret = ParseHttpRequest( e ) ;
                if( UNLIKELY(nret) )
                {
                        printf( "ParseHttpRequest failed[%d]\n" , nret );
                        DestroyHttpEnv( e );
                        return -1;
                }
        }

        DestroyHttpEnv( e );

        return 0;
}

int main( int argc , char *argv[] )
{
        if( argc == 1 + 1 )
        {
                return -press( atoi(argv[1]) );
        }
        else
        {
                printf( "USAGE : press count\n" );
                exit(9);
        }
}
```

picohttpparser 自带的压测代码 bench.c

```
/*
 * Copyright (c) 2009-2014 Kazuho Oku, Tokuhiro Matsuno, Daisuke Murase,
 *                         Shigeo Mitsunari
 *
 * The software is licensed under either the MIT License (below) or the Perl
 * license.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to
 * deal in the Software without restriction, including without limitation the
```

```
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include <assert.h>
#include <stdio.h>
#include "picohttpparser.h"

#define REQ
\
    "GET /wp-content/uploads/2010/03/hello-kitty-darth-vader-pink.jpg HTTP/1.1\r\n"
\
    "Host: www.kittyhell.com\r\n"
\
    "User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; ja-JP-mac; rv:1.9.2.3) Gecko/20100401
Firefox/3.6.3 "                    \
    "Pathtraq/0.9\r\n"
\
    "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
\
    "Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n"
\
    "Accept-Encoding: gzip,deflate\r\n"
\
    "Accept-Charset: Shift_JIS,utf-8;q=0.7,*;q=0.7\r\n"
\
    "Keep-Alive: 115\r\n"
\
    "Connection: keep-alive\r\n"
\
    "Cookie: wp_ozh_wsa_visits=2; wp_ozh_wsa_visit_lasttime=xxxxxxxxxx; "
\
```

```
    "__utma=xxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.xxxxxxxxxx.x; "
\

"__utmz=xxxxxxxxx.xxxxxxxxxx.x.x.utmccn=(referral)|utmcsr=reader.livedoor.com|utmcct=/reader/|utmcmd=ref
erral\r\n"                      \
    "\r\n"

int main(void)
{
    const char *method;
    size_t method_len;
    const char *path;
    size_t path_len;
    int minor_version;
    struct phr_header headers[32];
    size_t num_headers;
    int i, ret;

    for (i = 0; i < 10000000; i++) {
        num_headers = sizeof(headers) / sizeof(headers[0]);
        ret = phr_parse_request(REQ, sizeof(REQ) - 1, &method, &method_len, &path, &path_len,
&minor_version, headers, &num_headers,
                                        0);
        assert(ret == sizeof(REQ) - 1);
    }

    return 0;
}
```

5 开发参考

5.1 HTTP 环境

5.1.1 环境

CreateHttpEnv

原型：	struct HttpEnv *CreateHttpEnv();

说明：	创建 HTTP 环境
参数：	（无）
返回值：	HTTP 环境结构

ResetHttpEnv

原型：	void ResetHttpEnv(struct HttpEnv *e);
说明：	销毁 HTTP 环境重置 HTTP 环境
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	（无）

DestroyHttpEnv

原型：	void DestroyHttpEnv(struct HttpEnv *e);
说明：	销毁 HTTP 环境
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	（无）

5.1.2 环境属性

SetHttpTimeout

原型：	void SetHttpTimeout(struct HttpEnv *e , long timeout);
说明：	设置 HTTP 通讯接收超时时间。一个 HTTP 环境中只需设置一次，重置环境时同时重置为首次超时时间
参数：	struct HttpEnv *e　　HTTP 环境结构 long timeout　　通讯接收超时时间，单位：秒

返回值：	（无）

EnableHttpResponseCompressing

原型：	void EnableHttpResponseCompressing(struct HttpEnv *e , int enable_response_compressing);
说明：	激活 HTTP 自动压缩解压。目前只对客户端接收 HTTP 响应，服务端接收 HTTP 请求和发送 HTTP 响应时起作用
参数：	struct HttpEnv *e HTTP 环境结构 int enable_response_compressing 0:禁用 1:启用
返回值：	（无）

SetParserCustomIntData

原型：	void SetParserCustomIntData(struct HttpEnv *e , int i);
说明：	保存自定义数据到 HTTP 环境中
参数：	struct HttpEnv *e HTTP 环境结构 int i 整型数据
返回值：	（无）

GetParserCustomIntData

原型：	int GetParserCustomIntData(struct HttpEnv *e);
说明：	从 HTTP 环境中取出自定义数据
参数：	struct HttpEnv *e HTTP 环境结构
返回值：	整型数据

SetParserCustomPtrData

原型：	void SetParserCustomPtrData(struct HttpEnv *e , void *ptr);
说明：	保存自定义数据到 HTTP 环境中
参数：	struct HttpEnv *e　　HTTP 环境结构 char *ptr　　指针型数据
返回值：	（无）

GetParserCustomPtrData

原型：	void *GetParserCustomPtrData(struct HttpEnv *e);
说明：	从 HTTP 环境中取出自定义数据
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	指针型数据

5.1.3 全局参数

ResetAllHttpStatus

原型：	void ResetAllHttpStatus();
说明：	重置所有 HTTP 响应状态码及描述文本
参数：	（无）
返回值：	（无）

SetHttpStatus

原型：	void SetHttpStatus(int status_code , char *status_code_s , char

	*status_text);
说明：	设置 HTTP 响应状态码及描述文本。用于自定义 HTTP 状态码及描述文本
参数：	int status_code HTTP 状态码，如：404 char *status_code_s 字符串型 HTTP 状态码，如："404" char *status_text 描述文本，如："Not Found"
返回值：	（无）

SetHttpStatus

原型：	void SetHttpStatus(int status_code , char *status_code_s , char *status_text);
说明：	设置 HTTP 响应状态码及描述文本。用于自定义 HTTP 状态码及描述文本
参数：	int status_code HTTP 状态码，如：404 char *status_code_s 字符串型 HTTP 状态码，如："404" char *status_text 描述文本，如："Not Found"
返回值：	（无）

5.2 HTTP 通讯与解析

5.2.1 HTTP 客户端高层 API

RequestHttp

原型：	int RequestHttp(SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	发送 HTTP 请求，以及接收、解析 HTTP 响应。请求前必须在请求缓冲区内组织好 HTTP 请求数据

参数：	SOCKET sock	socket 描述字，HTTP 协议用
	SSL *ssl	ssl 结构指针，HTTPS 协议用
	struct HttpEnv *e	HTTP 环境结构
返回值：	0:成功 非 0:失败	

5.2.2 HTTP 客户端低层 API

SendHttpRequest

原型：	int SendHttpRequest(SOCKET sock , SSL *ssl , struct HttpEnv *e);	
说明：	发送 HTTP 请求。请求前必须在请求缓冲区内组织好 HTTP 请求数据	
参数：	SOCKET sock	socket 描述字，HTTP 协议用
	SSL *ssl	ssl 结构指针，HTTPS 协议用
	struct HttpEnv *e	HTTP 环境结构
返回值：	0:成功 非 0:失败	

ReceiveHttpResponse

原型：	int ReceiveHttpResponse(SOCKET sock , SSL *ssl , struct HttpEnv *e);	
说明：	接收、解析 HTTP 响应	
参数：	SOCKET sock	socket 描述字，HTTP 协议用
	SSL *ssl	ssl 结构指针，HTTPS 协议用
	struct HttpEnv *e	HTTP 环境结构
返回值：	0:成功 非 0:失败	

5.2.3 HTTP 客户端低层 API（非堵塞版本）

SendHttpRequestNonblock

原型：	int SendHttpRequestNonblock (SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	SendHttpRequest 的非堵塞版本
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构
返回值：	0:成功 FASTERHTTP_INFO_TCP_SEND_WOULDBLOCK:缓冲区内还有数据等待发送 非 0:失败

ReceiveHttpResponseNonblock

原型：	int ReceiveHttpResponseNonblock (SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	ReceiveHttpResponse 的非堵塞版本
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构
返回值：	0:成功 FASTERHTTP_INFO_NEED_MORE_HTTP_BUFFER:缓冲区内数据不完整 非 0:失败

5.2.4 HTTP 服务端高层 API

ResponseAllHttp

原型：	int ResponseAllHttp(SOCKET sock , SSL *ssl , struct HttpEnv *e , funcProcessHttpRequest *pfuncProcessHttpRequest , void *p);
说明：	接收、解析 HTTP 请求，调用回调函数处理之，然后发送 HTTP 响应，如果客户端要求连接保持（Connection: Keep-Alive），则迭代之。 回调函数原型：typedef int funcProcessHttpRequest(struct HttpEnv *e , void *p);
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构 funcProcessHttpRequest *pfuncProcessHttpRequest　　回调函数指针 void *p　　自定义数据，直接传递入回调函数
返回值：	0:成功 非 0:失败

5.2.5 HTTP 服务端低层 API

ReceiveHttpRequest

原型：	int ReceiveHttpRequest(SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	接收、解析 HTTP 请求
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构
返回值：	0:成功 FASTERHTTP_INFO_TCP_CLOSE:安全断开连接

	非 0:失败

FormatHttpResponseStartLine

原型：	int FormatHttpResponseStartLine(int status_code , struct HttpEnv *e , int fill_body_with_status_flag);
说明：	在 HTTP 响应缓冲区中组织首行数据
参数：	int status_code　　HTTP 响应状态码 struct HttpEnv *e　　HTTP 环境结构 int fill_body_with_status_flag　　1:同时组织 HTTP 体，当错误时使用 0:不组织 HTTP 体，默认
返回值：	0:成功 非 0:失败

SendHttpRequest

原型：	int SendHttpRequest(SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	发送 HTTP 响应
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构
返回值：	0:成功 非 0:失败

CheckHttpKeepAlive

原型：	int CheckHttpKeepAlive(struct HttpEnv *e);

说明：	检查本次请求是否需要保持连接
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	0:不保持连接 1:需要保持连接

5.2.6 HTTP 服务端低层 API（非堵塞版本）

ReceiveHttpRequestNonblock

原型：	int ReceiveHttpRequestNonblock (SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	ReceiveHttpRequest 的非堵塞版本
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构
返回值：	0:成功 FASTERHTTP_INFO_TCP_CLOSE:安全断开连接 FASTERHTTP_INFO_NEED_MORE_HTTP_BUFFER:缓冲区内数据不完整 非 0:失败

SendHttpRequestNonblock

原型：	int SendHttpRequestNonblock (SOCKET sock , SSL *ssl , struct HttpEnv *e);
说明：	SendHttpRequest 的非堵塞版本
参数：	SOCKET sock　　socket 描述字，HTTP 协议用 SSL *ssl　　ssl 结构指针，HTTPS 协议用 struct HttpEnv *e　　HTTP 环境结构

返回值：	0:成功 FASTERHTTP_INFO_TCP_SEND_WOULDBLOCK:缓冲区内还有数据等待发送 非 0:失败

5.3 HTTP 头与体信息

5.3.1 HTTP 首行头信息

GetHttpHeaderPtr_METHOD

原型：	char *GetHttpHeaderPtr_METHOD(struct HttpEnv *e , int *p_value_len);
说明：	得到 HTTP 请求头首行 METHOD 值
参数：	struct HttpEnv *e HTTP 环境结构 int *p_value_len 回传请求头首行头 METHOD 的值长度，如果置为 NULL 则不回传
返回值：	HTTP 请求头首行 METHOD 值地址

GetHttpHeaderLen_METHOD

原型：	int GetHttpHeaderLen_METHOD(struct HttpEnv *e);
说明：	得到 HTTP 请求头首行 METHOD 值长度
参数：	struct HttpEnv *e HTTP 环境结构
返回值：	HTTP 请求头首行 METHOD 值长度

GetHttpHeaderPtr_URI

原型：	char *GetHttpHeaderPtr_URI(struct HttpEnv *e , int *p_value_len);
说明：	得到 HTTP 请求头首行 URI 值
参数：	struct HttpEnv *e　　HTTP 环境结构 int *p_value_len 回传 HTTP 请求头首行 URI 的值长度，如果置为 NULL 则不回传
返回值：	HTTP 请求头首行 URI 值地址

GetHttpHeaderLen_URI

原型：	int GetHttpHeaderLen_URI(struct HttpEnv *e);
说明：	得到 HTTP 请求头首行 URI 值长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 请求头首行 URI 值长度

GetHttpHeaderPtr_VERSION

原型：	char *GetHttpHeaderPtr_VERSION(struct HttpEnv *e , int *p_value_len);
说明：	得到 HTTP 请求头首行 VERSION 值
参数：	struct HttpEnv *e　　HTTP 环境结构 int *p_value_len 回传 HTTP 请求头首行 VERSION 的值长度，如果置为 NULL 则不回传
返回值：	HTTP 请求头首行 VERSION 值地址

GetHttpHeaderLen_VERSION

原型：	int GetHttpHeaderLen_VERSION(struct HttpEnv *e);
说明：	得到 HTTP 首行请求请求头 VERSION 值长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 请求头首行 VERSION 值长度

GetHttpHeaderPtr_STATUSCODE

原型：	char *GetHttpHeaderPtr_STATUSCODE(struct HttpEnv *e , int *p_value_len);
说明：	得到 HTTP 响应头首行 STATUSCODE 值
参数：	struct HttpEnv *e　　HTTP 环境结构 int *p_value_len 回传 HTTP 响应头首行 STATUSCODE 的值长度，如果置为 NULL 则不回传
返回值：	HTTP 响应头首行 STATUSCODE 值地址

GetHttpHeaderLen_STATUSCODE

原型：	int GetHttpHeaderLen_STATUSCODE(struct HttpEnv *e);
说明：	得到 HTTP 响应头首行 STATUSCODE 值长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 响应头首行 STATUSCODE 值长度

GetHttpHeaderPtr_REASONPHRASE

原型：	char *GetHttpHeaderPtr_REASONPHRASE(struct HttpEnv *e , int

	*p_value_len);
说明：	得到 HTTP 响应头首行 REASONPHRASE 值
参数：	struct HttpEnv *e　　HTTP 环境结构 int *p_value_len 回传 HTTP 响应头首行 REASONPHRASE 的值长度，如果置为 NULL 则不回传
返回值：	HTTP 响应头首行 REASONPHRASE 值地址

GetHttpHeaderLen_REASONPHRASE

原型：	int GetHttpHeaderLen_REASONPHRASE(struct HttpEnv *e);
说明：	得到 HTTP 响应头首行 REASONPHRASE 值长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	响应头首行 REASONPHRASE 值长度

5.3.2 HTTP 头选项

QueryHttpHeaderPtr

原型：	char *QueryHttpHeaderPtr(struct HttpEnv *e , char *name , int *p_value_len);
说明：	得到 HTTP 头选项的值
参数：	struct HttpEnv *e　　HTTP 环境结构 char *name　　HTTP 头选项名 int *p_value_len 回传头选项的值长度。如果置为 NULL 则不回传
返回值：	HTTP 头选项的值地址

QueryHttpHeaderLen

原型：	int QueryHttpHeaderLen(struct HttpEnv *e , char *name);
说明：	得到 HTTP 头选项的值长度
参数：	struct HttpEnv *e　　HTTP 环境结构 char *name　　HTTP 头选项名
返回值：	HTTP 头选项的值长度

CountHttpHeaders

原型：	int CountHttpHeaders(struct HttpEnv *e);
说明：	得到 HTTP 头选项的数量
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 头选项的数量

TravelHttpHeaderPtr

原型：	struct HttpHeader *TravelHttpHeaderPtr(struct HttpEnv *e , struct HttpHeader *p_header);
说明：	遍历 HTTP 头选项
参数：	struct HttpEnv *e　　HTTP 环境结构 struct HttpHeader *p_header　　上一次遍历到的 HTTP 头选项结构。首次置为 NULL
返回值：	NULL:没有 HTTP 头选项或遍历结束了 非 NULL:本次遍历到 HTTP 头选项结构地址

GetHttpHeaderNamePtr

原型：	char *GetHttpHeaderNamePtr(struct HttpHeader *p_header , int *p_name_len);
说明：	得到 HTTP 头选项结构中的名字
参数：	struct HttpEnv *e HTTP 环境结构 int *p_name_len HTTP 头选项结构中的名字长度。如果置为 NULL 则不回传
返回值：	HTTP 头选项结构中的名字地址

GetHttpHeaderNameLen

原型：	int GetHttpHeaderNameLen(struct HttpHeader *p_header);
说明：	得到 HTTP 头选项结构中的名字长度
参数：	struct HttpEnv *e HTTP 环境结构
返回值：	HTTP 头选项结构中的名字长度

GetHttpHeaderValuePtr

原型：	char *GetHttpHeaderValuePtr(struct HttpHeader *p_header , int *p_value_len);
说明：	得到 HTTP 头选项结构中的值
参数：	struct HttpEnv *e HTTP 环境结构 int *p_value_len HTTP 头选项结构中的值长度。如果置为 NULL 则不回传
返回值：	HTTP 头选项结构中的值地址

GetHttpHeaderValueLen

原型：	int GetHttpHeaderNameLen(struct HttpHeader *p_header);
说明：	得到 HTTP 头选项结构中的值长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 头选项结构中的值长度

5.3.3 HTTP 体

GetHttpBodyPtr

原型：	char *GetHttpBodyPtr(struct HttpEnv *e , int *p_body_len);
说明：	得到 HTTP 体数据
参数：	struct HttpEnv *e　　HTTP 环境结构 int *p_body_len HTTP 体数据长度。如果置为 NULL 则不回传
返回值：	HTTP 体数据地址

GetHttpBodyLen

原型：	int GetHttpBodyLen(struct HttpEnv *e);
说明：	得到 HTTP 体数据长度
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 体数据长度

5.4 HTTP 缓冲区

5.4.1 得到 HTTP 缓冲区结构

GetHttpRequestBuffer

原型：	struct HttpBuffer *GetHttpRequestBuffer(struct HttpEnv *e);
说明：	得到 HTTP 请求缓冲区结构
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 请求缓冲区结构地址

GetHttpResponseBuffer

原型：	struct HttpBuffer *GetHttpResponseBuffer(struct HttpEnv *e);
说明：	得到 HTTP 响应缓冲区结构
参数：	struct HttpEnv *e　　HTTP 环境结构
返回值：	HTTP 响应缓冲区结构地址

5.4.2 得到 HTTP 缓冲区结构内信息

GetHttpBufferBase

原型：	char *GetHttpBufferBase(struct HttpBuffer *b , int *p_data_len);
说明：	得到 HTTP 缓冲区有效数据
参数：	struct HttpBuffer *b　HTTP 缓冲区结构 int *p_data_len　　HTTP 缓冲区有效数据长度。如果置为 NULL 则不回传
返回值：	HTTP 缓冲区有效数据地址

GetHttpBufferLength

原型：	int GetHttpBufferLength(struct HttpBuffer *b);
说明：	得到 HTTP 缓冲区有效数据长度
参数：	struct HttpBuffer *b HTTP 缓冲区结构
返回值：	HTTP 缓冲区有效数据长度

5.4.3 组织 HTTP 缓冲区数据

StrcpyHttpBuffer

原型：	int StrcpyHttpBuffer(struct HttpBuffer *b , char *str);
说明：	复制字符串覆盖到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *str 字符串
返回值：	0:成功 非 0:失败

StrcpyfHttpBuffer

原型：	int StrcpyfHttpBuffer(struct HttpBuffer *b , char *format , ...);
说明：	格式化字符串覆盖到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *format 格式化串（参考 sprintf） ... 参数列表（参考 sprintf）
返回值：	0:成功

	非 0:失败

StrcpyvHttpBuffer

原型：	int StrcpyvHttpBuffer(struct HttpBuffer *b , char *format , va_list valist);
说明：	格式化字符串覆盖到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *format 格式化串（参考 vsprintf） va_list valist 参数列表（参考 vsprintf）
返回值：	0:成功 非 0:失败

StrcatHttpBuffer

原型：	int StrcatHttpBuffer(struct HttpBuffer *b , char *str);
说明：	复制字符串追加到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *str 字符串
返回值：	0:成功 非 0:失败

StrcatfHttpBuffer

原型：	int StrcatfHttpBuffer(struct HttpBuffer *b , char *format , ...);
说明：	格式化字符串追加到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构

	char *format 格式化串（参考 sprintf） ... 参数列表（参考 sprintf）
返回值：	0:成功 非 0:失败

StrcatvHttpBuffer

原型：	int StrcpyvHttpBuffer(struct HttpBuffer *b , char *format , va_list valist);
说明：	格式化字符串追加到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *format 格式化串（参考 vsprintf） va_list valist 参数列表（参考 vsprintf）
返回值：	0:成功 非 0:失败

MemcatHttpBuffer

原型：	int MemcatHttpBuffer(struct HttpBuffer *b , char *base , int len);
说明：	复制二进制数据追加到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *base 源数据基地址（参考 memcpy） int len 源数据有效长度（参考 memcpy）
返回值：	0:成功 非 0:失败

StrcatHttpBufferFromFile

原型：	int StrcatHttpBufferFromFile(struct HttpBuffer *b , char *pathfilename , int *p_filesize);
说明：	复制文件数据追加到 HTTP 缓冲区
参数：	struct HttpBuffer *b HTTP 缓冲区结构 char *pathfilename 带路径的文件名 int *p_filesize 文件数据截取长度。如果置为 NULL 则读取整个文件
返回值：	0:成功 非 0:失败

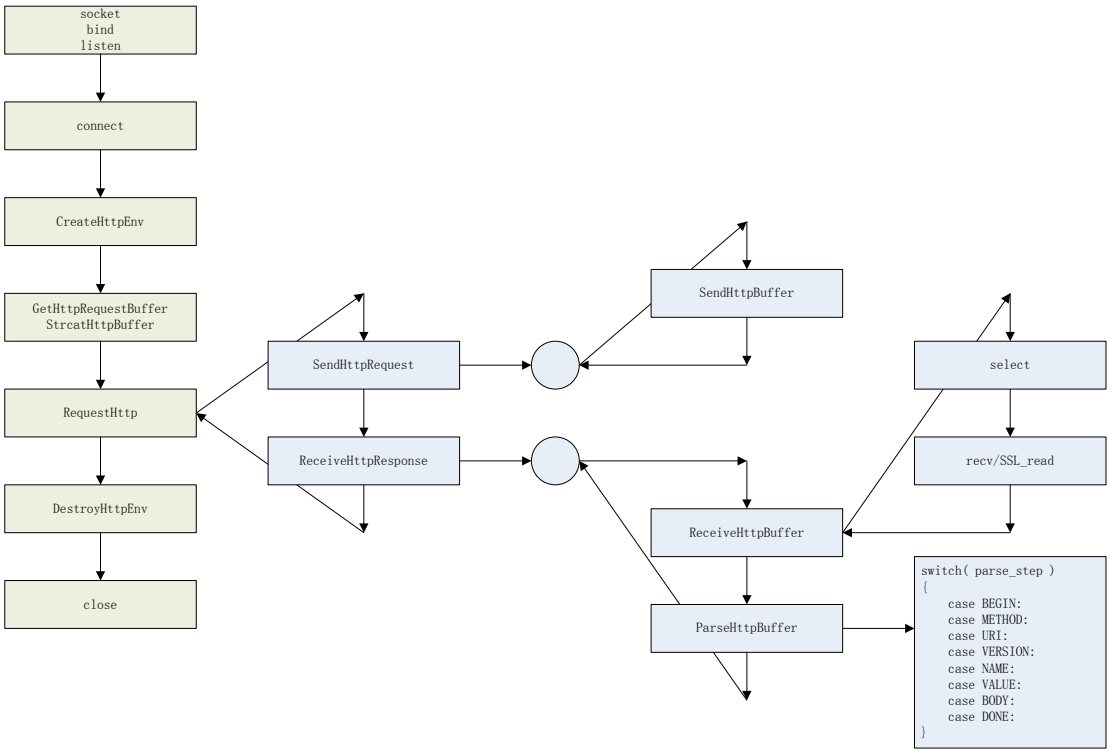
5.5 工具函数

SetHttpNonblock

原型：	void SetHttpNonblock(int sock);
说明：	设置 socket 描述字为非堵塞
参数：	int sock socket 描述字
返回值：	（无）

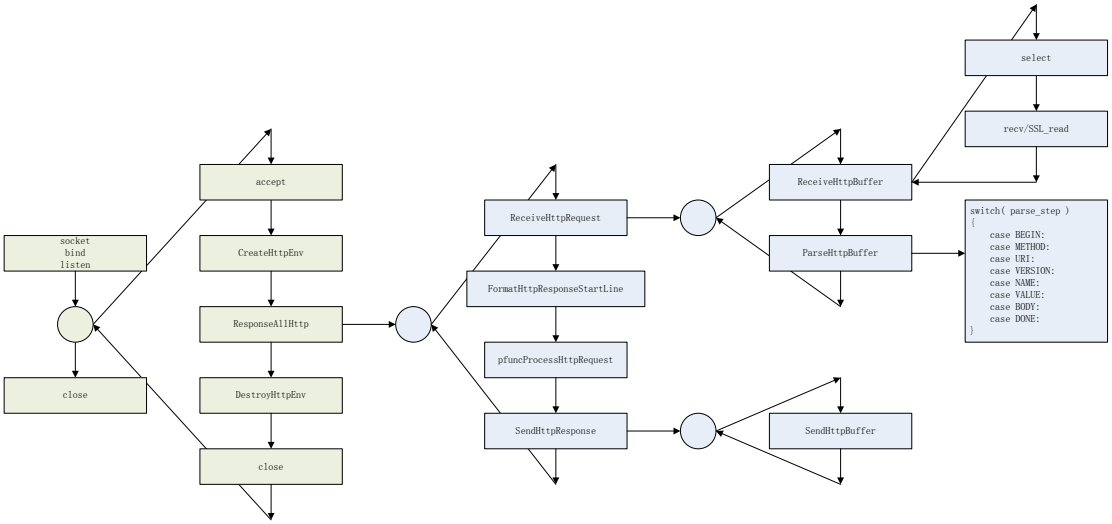
6 内部实现

6.1 客户端 API 内部调用关系



以上为客户端（堵塞模式）API 内部调用关系图，土黄色为应用层流程，蓝色为 fasterhttp 内部流程。可见 RequestHttp 调用了 SendHttpRequest 发送 HTTP 请求，再调用了 ReceiveHttpResponse 接收、解析 HTTP 响应。SendHttpRequest 调用了内部公共函数 SendHttpBuffer 发送缓冲区数据到 TCP。ReceiveHttpResponse 内部循环调用 ReceiveHttpBuffer 和 ParseHttpBuffer，直到 HTTP 响应数据接收完整且解析完成。

6.2 服务端 API 内部调用关系



以上为服务端（堵塞模式）API 内部调用关系，土黄色为应用层流程，蓝色为 fasterhttp 内部流程。可见 ResponseAllHttp 内部调用 ReceiveHttpRequest 接收、解析 HTTP 请求，再调用 FormatHttpResponseStartLine 默认组织响应首行，然后调用 pfuncProcessHttpRequest 处理 HTTP 请求、生成 HTTP 响应，如果出错则再次调用 FormatHttpResponseStartLine 组织带错误码的响应首行，最后调用 SendHttpResponse 发送 HTTP 响应，如果客户端要求连接保持则再原连接上迭代。ReceiveHttpRequest 内部循环调用 ReceiveHttpBuffer 和 ParseHttpBuffer，直到 HTTP 请求数据接收完整且解析完成。SendHttpResponse 调用了内部公共函数 SendHttpBuffer 发送缓冲区数据到 TCP。