听风流过

【转】Zlib库的安装与使用

在实际应用中经常会遇到要压缩数据的问题，常见的压缩格式有zip和rar,而 Linux下那就更多了,bz2,gz,xz什么的都有，单单Linux下的解压和压缩命令就有好多呢？没有什么好不好的。查了资料，应该是zlib这个 比较简单好用。应用也广，所以就准备用这个了。

下载Zlib库，地址: http://zlib.net /zlib128.zip 用wget下载，然后再用unzip解压一下，然后就像一般软件一样 ./configure && make && make install .(注意要root权限)

下面这个是安装信息

```
cp libz.a  / usr / local / lib
chmod  644   / usr / local / lib / libz.a
cp libz.so. 1.2 . 8   / usr / local / lib
chmod  755   / usr / local / lib / libz.so. 1.2 . 8
cp zlib. 3   / usr / local / share / man / man3
chmod  644   / usr / local / share / man / man3 / zlib. 3
cp zlib.pc  / usr / local / lib / pkgconfig
chmod  644   / usr / local / lib / pkgconfig / zlib.pc
cp zlib.h zconf.h  / usr / local / include
chmod  644   / usr / local / include / zlib.h  / usr / local / include / zconf.h
```

写一个简单的例子测试一下，注意编译的时候要加入 -lz 这个库

```
#include  < stdio.h >
#include  < zlib.h >

int  main( int  argc, char   ** args)
{
    /* 原始数据 */
   unsigned  char  strsrc[] = " 这些是测试数据。123456789 abcdefghigklmnopqrstuvwxyz\n\t\0abcdefghijklmnopqrstuvwxyz\n " ;  // 包含\0字符
   unsigned  char  buf[ 1024 ] = { 0 };
   unsigned  char  strdst[ 1024 ] = { 0 };
   unsigned  long  srclen = sizeof (strsrc);
   unsigned  long  buflen = sizeof (buf);
   unsigned  long  dstlen = sizeof (strdst);
    int  i;
   FILE  *  fp;

   printf( " 源串: " );
    for (i = 0 ;i < srclen; ++ i)
   {
      printf( " %c " ,strsrc[i]);
   }
   printf( " 原串长度为:%ld\n " ,srclen);

   printf( " 字符串预计算长度为:%ld\n " ,compressBound(srclen));
    // 压缩
   compress(buf, & buflen,strsrc,srclen);
   printf( " 压缩后实际长度为:%ld\n " ,buflen);
    // 解压缩
   uncompress(strdst, & dstlen,buf,buflen);

   printf( " 目的串: " );
    for (i = 0 ;i < dstlen; ++ i)
   {
      printf( " %c " ,strdst[i]);
   }

    return   0 ;
}
```

各个API

```
1 //把源缓冲压缩成到目的缓冲,一个函数就完成了
2 int compress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);
3
4 //功能和compress函数一样,多了一个参数可以指定压缩质量和压缩数度之间的关系(0-9)。要想得到高的压缩比就要多花时间
5 int compress2 (Bytef *dest, uLongf *destLen,const Bytef *source, uLong sourceLen,int level);
6
7 //计算需要的缓冲区长度，假设你在压缩之前就想知道你的产度为 sourcelen 的数据压缩后有多大，可调用这个函数计算一下,这个函数并不能得到精确的结果,但是它可以保证实际输出长度肯定小于它计算出来的长度
8 uLong compressBound (uLong sourceLen);
9
10 //解压缩
11 int uncompress (Bytef *dest, uLongf *destLen,const Bytef *source, uLong sourceLen);
```

处理gz后缀的压缩文件。

压缩字符串到test.gz

```
1 #include <stdio.h>
2 #include <zlib.h>
3
4 int main(int argc,char **args)
5 {
6    gzFile file;
7    char str[]="testtest";
8    file=gzopen("test.gz","wb");
9    if(NULL==file)
10       perror("Can't open file");
11    gzsetparams(file,2,0);
12    gzwrite(file,str,sizeof(str));
13    gzclose(file);
14    return 0;
15 }
```

解压test.gz

```
1 #include <stdio.h>
2 #include <zlib.h>
3
4 int main(int argc,char **args)
5 {
6    gzFile file;
7    char str[64]={0};
8    file=gzopen("test.gz","rb");
9    if(NULL==file)
10       perror("Can't open file");
11    gzread(file,str,10);
12    printf("从文件中读取到的字符:%s.\n",str);
13    gzclose(file);
14    return 0;
15 }
```

对于test.gz这个文件如果不知道是否正确，可以使用系统命令zcat进 行查看。还有一个问题就是个压缩包只能解压出一个文件,是不可能存在多个文件的。这是GZIP的特性决定了的。通常，都是把多个文件用tar先压成一个包,然后在gzip。这就是为什么下载的软件源码包大多都是 .tar.gz 这样的格式。

对于gz文件处理的API

```
//注意下面介绍到的处理函数，跟一般文件的处理函数是相似的，效果也是差不多的。
typedef voidp gzFile;

//打开一个gzip文件进行读/写,mode和fopen("rb"或" wb")一样.也可以包括压缩级别如:"wb9",或着一个策略"f"作为过滤数据"wb6f", "h"是为了"huffman" 压缩,如:"wb1h".gzopen用于读一个没有gzip格式的文件.gzread直接从没有解压缩的文件中读数据.如果文件不能被打开或是没有足够的内存,gzopen将返回NULL.
gzFile gzopen(const char *path,const char *mode);

//根据文件描述符打开一个gz文件
gzFile gzdopen (int fd, const char *mode);

//动态更新压缩等级和压缩格式 成功返回Z_OK 否则返回Z_STREAM_ERROR
int gzsetparams (gzFile file, int level, int strategy);

//读取所给的个数len字节。从压缩文件中读取解压后的字符的个数。如果file不是gzip格式，那么就读取实际的字节个数，而不是解压后的实际个数。成功返回所读取的个数，0表示文件结束，-1表示错误。
//英文原文 Reads the given number of uncompressed bytes from the compressed file.If the input file was not in gzip format, gzread copies the given number of bytes into the buffer.
int gzread (gzFile file,voidp buf, unsigned int len);

//写入所给长度的buf字符到file中 成功返回所写入的字符个数，0表示写入失败
int gzwrite(gzFile file, voidp buf,unsigned int len);

//可以看到下面介绍到的所有函数都跟我们处理普通的文件一样，把函数的gz换成f，是不是就熟悉了许多？大概就是这样吧
int gzprintf(gzFile file,const char *format, …);
int gzputs(gzFile file,const char *s);
char * gzgets(gzFile file, char *buf,int len);
int gzputc(gzFile file,int c);
int gzgetc(gzFile file);
int gzungetc(int c,gzFile file);
int gzflush (gzFile file,int flush);
int gzseek (gzFile file,z_off_t offset, int whence);
int gzrewind(gzFile file);
z_off_t gztell(gzFile file);
int gzeof(gzFile file);
int gzdirect(gzFile file);
int gzclose(gzFile file);
const char * gzerror(gzFile file, int *errnum);
void gzclearerr(gzFile file);
```

计算校验码

```
uLong adler32 (uLong adler,const Bytef *buf, uInt len);
//使用方法如下
uLong adler=adler32(0L,Z_NULL,0);
while(read_buffer(buffer, length) !=EOF)
{
   adler=adler32(adler,buffer,length);
}
if(adler != original_adler)
   error();

uLong crc32 (uLong crc,const Bytef *buf,uInt len);
//使用方法如下
uLong crc = crc32(0L,Z_NULL,0);
while(read_buffer(buffer,length)!=EOF)
{
   crc=crc32(crc,buffer,length);
}
if(crc != original_crc)
   error();
```

最后是z_stream这个结构了

```
typedef struct z_stream_s {
   Bytef    *next_in;  /* next input byte */
   uInt     avail_in;  /* number of bytes available at next_in */
   uLong    total_in;  /* total nb of input bytes read so far */

   Bytef    *next_out; /* next output byte should be put there */
   uInt     avail_out; /* remaining free space at next_out */
   uLong    total_out; /* total nb of bytes output so far */

   char     *msg;      /* last error message, NULL if no error */
   struct internal_state FAR *state; /* not visible by applications */

   alloc_func zalloc;  /* used to allocate the internal state */
   free_func  zfree;   /* used to free the internal state */
   voidpf     opaque;  /* private data object passed to zalloc and zfree */

   int     data_type;  /* best guess about the data type: binary or text */
   uLong   adler;      /* adler32 value of the uncompressed data */
   uLong   reserved;   /* reserved for future use */
} z_stream;
```

```
deflateInit() + deflate() + deflateEnd()
//3个函数结合使用完成压缩功能,具体用法看 example.c 的 test_deflate()函数. 其实compress() 函数内部就是用这3个函数实现的

inflateInit() + inflate() + inflateEnd()
//上面类似,完成解压缩功能.
```

下面给出一个example方面查看，了解函数的用法，不过一般应用程序用到上面的函数即可。

```
/* zpipe.c: example of proper use of zlib's inflate() and deflate()
   Not copyrighted -- provided to the public domain
   Version 1.4  11 December 2005  Mark Adler */

/* Version history:
   1.0  30 Oct 2004  First version
   1.1   8 Nov 2004  Add void casting for unused return values
                     Use switch statement for inflate() return values
   1.2   9 Nov 2004  Add assertions to document zlib guarantees
   1.3   6 Apr 2005  Remove incorrect assertion in inf()
   1.4  11 Dec 2005  Add hack to avoid MSDOS end-of-line conversions
                     Avoid some compiler warnings for input and output buffers
 */

#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "zlib.h"

#if defined(MSDOS) || defined(OS2) || defined(WIN32) || defined(__CYGWIN__)
#  include <fcntl.h>
#  include <io.h>
#  define SET_BINARY_MODE(file) setmode(fileno(file), O_BINARY)
#else
#  define SET_BINARY_MODE(file)
#endif

#define CHUNK 16384

/* Compress from file source to file dest until EOF on source.
   def() returns Z_OK on success, Z_MEM_ERROR if memory could not be
   allocated for processing, Z_STREAM_ERROR if an invalid compression
   level is supplied, Z_VERSION_ERROR if the version of zlib.h and the
   version of the library linked do not match, or Z_ERRNO if there is
   an error reading or writing the files. */
int def(FILE *source, FILE *dest, int level)
{
```

```c
    int ret, flush;
    unsigned have;
    z_stream strm;
    unsigned char in[CHUNK];
    unsigned char out[CHUNK];

    /* allocate deflate state */
    strm.zalloc = Z_NULL;
    strm.zfree = Z_NULL;
    strm.opaque = Z_NULL;
    ret = deflateInit(&strm, level);
    if (ret != Z_OK)
        return ret;

    /* compress until end of file */
    do {
        strm.avail_in = fread(in, 1, CHUNK, source);
        if (ferror(source)) {
            (void)deflateEnd(&strm);
            return Z_ERRNO;
        }
        flush = feof(source) ? Z_FINISH : Z_NO_FLUSH;
        strm.next_in = in;

        /* run deflate() on input until output buffer not full, finish
           compression if all of source has been read in */
        do {
            strm.avail_out = CHUNK;
            strm.next_out = out;
            ret = deflate(&strm, flush);    /* no bad return value */
            assert(ret != Z_STREAM_ERROR);  /* state not clobbered */
            have = CHUNK - strm.avail_out;
            if (fwrite(out, 1, have, dest) != have || ferror(dest)) {
                (void)deflateEnd(&strm);
                return Z_ERRNO;
            }
        } while (strm.avail_out == 0);
        assert(strm.avail_in == 0);     /* all input will be used */

        /* done when last data in file processed */
    } while (flush != Z_FINISH);
    assert(ret == Z_STREAM_END);        /* stream will be complete */

    /* clean up and return */
    (void)deflateEnd(&strm);
    return Z_OK;
}

/* Decompress from file source to file dest until stream ends or EOF.
   inf() returns Z_OK on success, Z_MEM_ERROR if memory could not be
   allocated for processing, Z_DATA_ERROR if the deflate data is
   invalid or incomplete, Z_VERSION_ERROR if the version of zlib.h and
   the version of the library linked do not match, or Z_ERRNO if there
   is an error reading or writing the files. */
int inf(FILE *source, FILE *dest)
{
    int ret;
    unsigned have;
    z_stream strm;
    unsigned char in[CHUNK];
    unsigned char out[CHUNK];

    /* allocate inflate state */
    strm.zalloc = Z_NULL;
    strm.zfree = Z_NULL;
    strm.opaque = Z_NULL;
    strm.avail_in = 0;
    strm.next_in = Z_NULL;
    ret = inflateInit(&strm);
    if (ret != Z_OK)
        return ret;

    /* decompress until deflate stream ends or end of file */
    do {
        strm.avail_in = fread(in, 1, CHUNK, source);
        if (ferror(source)) {
            (void)inflateEnd(&strm);
            return Z_ERRNO;
        }
        if (strm.avail_in == 0)
            break;
        strm.next_in = in;

        /* run inflate() on input until output buffer not full */
        do {
            strm.avail_out = CHUNK;//CHUNK=128K
            strm.next_out = out;
            ret = inflate(&strm, Z_NO_FLUSH);
            assert(ret != Z_STREAM_ERROR);  /* state not clobbered */
            switch (ret) {
            case Z_NEED_DICT:
                ret = Z_DATA_ERROR;     /* and fall through */
            case Z_DATA_ERROR:
            case Z_MEM_ERROR:
                (void)inflateEnd(&strm);
                return ret;
            }
            have = CHUNK - strm.avail_out;
            if (fwrite(out, 1, have, dest) != have || ferror(dest)) {
                (void)inflateEnd(&strm);
                return Z_ERRNO;
            }
        } while (strm.avail_out == 0);

        /* done when inflate() says it's done */
    } while (ret != Z_STREAM_END);

    /* clean up and return */
    (void)inflateEnd(&strm);
    return ret == Z_STREAM_END ? Z_OK : Z_DATA_ERROR;
}

/* report a zlib or i/o error */
void zerr(int ret)
{
    fputs("zpipe: ", stderr);
    switch (ret) {
    case Z_ERRNO:
        if (ferror(stdin))
            fputs("error reading stdin\n", stderr);
        if (ferror(stdout))
            fputs("error writing stdout\n", stderr);
        break;
    case Z_STREAM_ERROR:
        fputs("invalid compression level\n", stderr);
        break;
    case Z_DATA_ERROR:
        fputs("invalid or incomplete deflate data\n", stderr);
        break;
```

```
    case Z_MEM_ERROR:
        fputs("out of memory\n", stderr);
        break;
    case Z_VERSION_ERROR:
        fputs("zlib version mismatch!\n", stderr);
    }
}

/* compress or decompress from stdin to stdout */
int main(int argc, char **argv)
{
    int ret;

    /* avoid end-of-line conversions */
    SET_BINARY_MODE(stdin);
    SET_BINARY_MODE(stdout);

    /* do compression if no arguments */
    if (argc == 1) {
        ret = def(stdin, stdout, Z_DEFAULT_COMPRESSION);
        if (ret != Z_OK)
            zerr(ret);
        return ret;
    }

    /* do decompression if -d specified */
    else if (argc == 2 && strcmp(argv[1], "-d") == 0) {
        ret = inf(stdin, stdout);
        if (ret != Z_OK)
            zerr(ret);
        return ret;
    }

    /* otherwise, report usage */
    else {
        fputs("zpipe usage: zpipe [-d] < source > dest\n", stderr);
        return 1;
    }
}
```

接下来给出一个实例来了解一下gzip。

Web服务器实现gzip压缩发送

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>
#include <zlib.h>
#include <zconf.h>

void app_exit();
int socket_listen(u_short port);
void send_http_head(int clifd);
void put_long (unsigned char *string, unsigned long x);
int gzip_http_buffer(int clifd,char *msg,int len);


int sockfd;
int main(int argc,char **args)
{
    struct sockaddr_in cli_sin;
    socklen_t cli_len=sizeof(cli_sin);
    int clifd;
    char buf[4096];
    char msg[4096]="<br><br><h1>Reage Web Server gzip support text!</h1><br/><h1><a href = \"http://wunaozai.cnblogs.com\">Reage blog</a>";

    signal(SIGINT,app_exit);
    sockfd=socket_listen(8080);
    while(1)
    {
        clifd=accept(sockfd,(struct sockaddr *)&cli_sin,&cli_len);
        printf("连接进来的IP:%s:%u\n",inet_ntoa(cli_sin.sin_addr),ntohs(cli_sin.sin_port));
        read(clifd,buf,4096);
        printf("%s\n",buf);
        send_http_head(clifd);
        gzip_http_buffer(clifd,msg,strlen(msg));

        close(clifd);
    }
    close(sockfd);
    return 0;
}

void put_long (unsigned char *string, unsigned long x)//对于gzip后面的两个字节进行位填充,这里应该是处理大端与小端
{
    string[0] = (x & 0xff);
    string[1] = ((x >> 8) & 0xff) ;
    string[2] = ((x >> 16) & 0xff) ;
    string[3] = ((x >> 24) & 0xff);
}
/*
 * 对要发送的msg里面的字符进行压缩发送
 * gzip格式: http://www.cnblogs.com/witxjp/archive/2003/12/17/1986210.html
 * */
static const char gzip_header[10]={0x1f,0x8b,0x08,0,0,0,0,0,0,0x03};
int gzip_http_buffer(int clifd,char *msg,int len)
{
    z_stream stream;
    int ret,flush;
    char in[4096];//存放输入的数据
    char send[4096+18];//存放压缩过后的数据
    unsigned int have;
    int tmp;
    memcpy(send,gzip_header,10);
    memset(in,0,len);
    stream.zalloc=Z_NULL;
    stream.zfree=Z_NULL;
    stream.opaque=Z_NULL;
    stream.avail_in=0;
    stream.next_in=Z_NULL;
    memcpy(in,msg,len);
    //压缩初始化
    tmp=deflateInit2(&stream,
            Z_DEFAULT_COMPRESSION,//压缩级别，从0-9
            Z_DEFLATED,//压缩方式
            -MAX_WBITS,
            8,
            Z_DEFAULT_STRATEGY);
    if(Z_OK!=tmp)
    {
        perror("deflateInit2 Error");
        return 0;
```

```
    }
    stream.avail_in = len; //要压缩数据的长度
    stream.next_in = in;    //要压缩数据的首地址
    stream.avail_out = 4096;  //可存放的最大输出结果的长多。就是压缩后数据的最大长度
    stream.next_out = send + 10; //存放压缩数据的开始位置，send前十个字节用来放头部
    ret = deflate (&stream,Z_FINISH); //压缩
    switch(ret)
    {
        case Z_NEED_DICT:
            ret=Z_DATA_ERROR;
        case Z_DATA_ERROR:
        case Z_MEM_ERROR:
            (void)inflateEnd(&stream);
            return ret;
    }
    have = 4096 - stream.avail_out;
    unsigned crc = crc32(0L, in, len);
    char * tail = send + 10 + have;
    put_long (tail, crc);
    put_long (tail + 4, len);
    write (clifd, send, have + 18);
    deflateEnd (&stream);
    return 1;
}

void app_exit()
{
    signal(SIGINT,SIG_DFL);
    close(sockfd);
    exit(0);
}

/*
 * 发送一个HTTP头
 * */
void send_http_head(int clifd)
{
    char buf[4096];
    memset(buf,0,sizeof(buf));
    sprintf(buf,"HTTP/1.1 200 OK\r\n");
    sprintf(buf,"%sServer:wunaozai.cnblogs.com\r\n",buf);
    sprintf(buf,"%sContent-Encoding: gzip\r\n",buf);//告诉浏览器，我接下来发送的数据是经过gzip压缩过的
    sprintf(buf,"%sContent-Type: text/html\r\n\r\n",buf);
    write(clifd,buf,strlen(buf));
}

/*
 * 开启服务器监听
 * port:服务器端口
 * 成功返回套接字标识
 * */
int socket_listen(u_short port)
{
    struct sockaddr_in sin;
    int on;
    int httpd=socket(PF_INET,SOCK_STREAM,0);
    if(httpd==-1)
        perror("Fail to Socket");
    //init sockaddr_in
    sin.sin_family=AF_INET;
    sin.sin_port=htons(port);
    sin.sin_addr.s_addr=htonl(INADDR_ANY);
    bzero(&(sin.sin_zero),8);
    setsockopt(httpd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
    if(bind(httpd,(struct sockaddr *)&sin,sizeof(struct sockaddr))==-1)
        perror("Fail to bind");
    //如果port指定为零那么就随机打开一个端口
    if(port==0)
    {
        socklen_t len=sizeof(sin);
        if(getsockname(httpd,(struct sockaddr *)&sin,&len)==-1)
            perror("Fail to getsockname");
        port=ntohs(sin.sin_port);
    }
    if(listen(httpd,100)<0)
        perror("Fail to listen");
    printf("打开端口:%u\n",port);
    return httpd;
}
```

参考资料: http://blog.csdn.net/reage11/article/details/8517631

: http://www.cppblog.com/Streamlet/archive/2010/09/22/127368.aspx

: http://blog.csdn.net/htttw/article/details/7616124

: http://www.cppblog.com/woaidongmao/archive/2009/09/07/95495.html

: http://blog.csdn.net/zhoudaxia/article/details/8039519

:http://www.cnblogs.com/wunaozai/p/3960494.html

posted on 2017-04-11 16:30 听风 阅读(563) 评论(0) 编辑 收藏 所属分类: C++