转载 xiaoyeyopulei 2012-08-30 09:37:05 2681 收藏 1
分类专栏： zlib 文章标签： compression stream dictionary output file buffer

Zlib函数列表和使用手册 http://blog.sina.com.cn/s/blog_956a84ad0100zeyx.html

在*www.zlib.net* 得到资料.自己整理了下,翻译了一部分. 整理成两部分,第一部分是提炼出来的函数列表.第二部分是原来的使用手册+自己翻译的.是用来自己用的.如果能给别人带来方便,也是意外的惊喜阿 ,哈哈.

---------- *First Part* ----------

***Zlib*函数列表**

实用函数

*int compress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);*
*int compress2 (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level);*
*int uncompress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);*
typedef voidp gzFile;
*gzFile gzopen (const char *path, const char *mode);*
*gzFile gzdopen (int fd, const char *mode);*
int gzsetparams (gzFile file, int level, int strategy);
int gzread (gzFile file, voidp buf, unsigned len);
int gzwrite (gzFile file, const voidp buf, unsigned len);
*int VA 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 gzflush (gzFile file, int flush);
z_off_t gzseek (gzFile file, z_off_t offset, int whence);
z_off_t gztell (gzFile file);
int gzrewind (gzFile file);
int gzeof (gzFile file);
int gzclose (gzFile file);
*const char * gzerror (gzFile file, int *errnum);*

基本函数：

*const char * zlibVersion (void);*
int deflateInit (z_streamp strm, int level);
int deflate (z_streamp strm, int flush);
int deflateEnd (z_streamp strm);
int inflateInit (z_streamp strm);
int inflate (z_streamp strm, int flush);
int inflateEnd (z_streamp strm);

高级函数：

int deflateInit2 (z_streamp strm,
*int deflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
int deflateCopy (z_streamp dest, z_streamp source);
int deflateReset (z_streamp strm);
int deflateParams (z_streamp strm, int level, int strategy);
int inflateInit2 (z_streamp strm, int windowBits);
*int inflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
int inflateSync (z_streamp strm);
int inflateReset (z_streamp strm);

校验函数

*uLong adler32 (uLong adler, const Bytef *buf, uInt len);*
*uLong crc32 (uLong crc, const Bytef *buf, uInt len);*
struct z_stream_s
typedef struct z_stream_s {
*Bytef *next_in;*
uInt avail_in;
uLong total_in;
*Bytef *next_out;*
uInt avail_out;
uLong total_out;
*char *msg;*
*struct internal_state FAR *state;*
alloc_func zalloc;
free_func zfree;
voidpf opaque;
int data_type;
uLong adler;
uLong reserved;
} z_stream ;
*typedef z_stream FAR * z_streamp;*

常量:

#define Z_NO_FLUSH 0
#define Z_PARTIAL_FLUSH 1
#define Z_SYNC_FLUSH 2
#define Z_FULL_FLUSH 3
#define Z_FINISH 4
#define Z_OK 0
#define Z_STREAM_END 1
#define Z_NEED_DICT 2
#define Z_ERRNO (-1)

分类专栏
linux内核 2篇
c 16篇
gsoap 4篇
gdb 2篇
xml 1篇
java 2篇
mysql 2篇
zlib 1篇
好玩 2篇
web 开发 5篇
linux多线程 10篇
Java 设计模式 3篇
Redis 1篇
算法 7篇

#define Z_STREAM_ERROR (-2)

#define Z_DATA_ERROR (-3)

#define Z_MEM_ERROR (-4)

#define Z_BUF_ERROR (-5)

#define Z_VERSION_ERROR (-6)

#define Z_NO_COMPRESSION 0

#define Z_BEST_SPEED 1

#define Z_BEST_COMPRESSION 9

#define Z_DEFAULT_COMPRESSION (-1)

#define Z_FILTERED 1

#define Z_HUFFMAN_ONLY 2

#define Z_DEFAULT_STRATEGY 0

#define Z_BINARY 0

#define Z_ASCII 1

#define Z_UNKNOWN 2

#define Z_DEFLATED 8

#define Z_NULL 0

#define zlib_version zlibVersion()

另外一些函数:

const char * zError (int err);

int inflateSyncPoint (z_streamp z);

const uLongf * get_crc_table (void);

----------SecondPart ----------

zlib 1.1.4 手册

目录

序言

'zlib'常用的流行的压缩库 。版本:1.1.4。

介绍

zlib压缩库提供内存内压缩/解压缩函数。包括解压验证。

这个版本只有一种压缩方式, 但是以后其他的算法也会被加入进来, 并且接口是一样的。

如果缓存区足够大, 压缩被一次完成, 否则就重复调用压缩。在后一种情况, 程序必须在每次调用时提供更多的输入或更多输出空间。

本压缩库也支持gzip(.gz)格式的读写操作。接口也和stdio相似。

本压缩库不安装任何信号处理, 解码器检查压缩数据的一致性, 所以, 本压缩库决不会使输入崩溃。

实用函数

以下实用函数的实现建立在basic stream-oriented 函数上。

为了了简化接口, 设置了一些默认选项(压缩级别, 内存使用, 标准内存分配器功能)这些实用函数的源代码很容易被修改, 如果你要实现有些特殊选项。

函数列表:

int compress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);

int compress2 (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level);

int uncompress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);

typedef voidp gzFile;

gzFile gzopen (const char *path, const char *mode);

gzFile gzdopen (int fd, const char *mode);

int gzsetparams (gzFile file, int level, int strategy);

int gzread (gzFile file, voidp buf, unsigned len);

int gzwrite (gzFile file, const voidp buf, unsigned len);

int VA 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 gzflush (gzFile file, int flush);

z_off_t gzseek (gzFile file, z_off_t offset, int whence);

z_off_t gztell (gzFile file);

int gzrewind (gzFile file);

int gzeof (gzFile file);

int gzclose (gzFile file);

const char * gzerror (gzFile file, int *errnum);

函数说明:

int compress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);

压缩source buffer到destination buffer。

sourceLen是source buffer的长度(byte)。

destLen是destination buffer的总共长度(byte)。调用前 destLen的长度必须至少sourceLen长度的0.1%再加上12个byte.调用后。destLen是实际的compressed buffer长度。

如果输入文件是mmap'ed, 这个函数可以用于压缩整个文件。

如果压缩成功返回Z_OK, 如果没有足够的内存返回Z_MEM_ERROR,如果没有足够的空间输出文件返回Z_BUF_ERROR.

int compress2 (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level);

压缩source buffer到destination buffer。

参数级level和defalteInit一样。

destLen是destination buffer的总共长度(byte)。调用前 destLen的长度必须至少sourceLen长度的0.1%再加上12个byte.调用后。destLen是实际的compressed buffer长度。

如果压缩成功返回Z_OK, 如果没有足够的内存返回Z_MEM_ERROR,如果没有足够的空间输出文件返回Z_BUF_ERROR.

如果level是无效的, 返回Z_STREAM_ERROR.

*int uncompress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen);*
Decompresses the source buffer into the destination buffer. sourceLen is the byte length of the source buffer. Upon entry, destLen is the total size of the destination buffer, which must be large enough to hold the entire uncompressed data. (The size of the uncompressed data must have been saved previously by the compressor and transmitted to the decompressor by some mechanism outside the scope of this compression library.) Upon exit, destLen is the actual size of the compressed buffer.
This function can be used to decompress a whole file at once if the input file is mmap'ed.
uncompress returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_BUF_ERROR if there was not enough room in the output buffer, or Z_DATA_ERROR if the input data was corrupted.

typedef voidp gzFile;
*gzFile gzopen (const char *path, const char *mode);*
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 gzdopen (int fd, const char *mode);*
gzdopen() associates a gzFile with the file descriptor fd. File descriptors are obtained from calls like open, dup, creat, pipe or fileno (in the file has been previously opened with fopen). The mode parameter is as in gzopen.
The next call of gzclose on the returned gzFile will also close the file descriptor fd, just like fclose(fdopen(fd), mode) closes the file descriptor fd. If you want to keep fd open, use gzdopen(dup(fd), mode).
gzdopen returns NULL if there was insufficient memory to allocate the (de)compression state.

int gzsetparams (gzFile file, int level, int strategy);
Dynamically update the compression level or strategy. See the description of deflateInit2 for the meaning of these parameters.
gzsetparams returns Z_OK if success, or Z_STREAM_ERROR if the file was not opened for writing.

int gzread (gzFile file, voidp buf, unsigned len);
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.
gzread returns the number of uncompressed bytes actually read (0 for end of file, -1 for error).

int gzwrite (gzFile file, const voidp buf, unsigned len);
Writes the given number of uncompressed bytes into the compressed file. gzwrite returns the number of uncompressed bytes actually written (0 in case of error).
*int VA gzprintf (gzFile file, const char *format, ...);*
Converts, formats, and writes the args to the compressed file under control of the format string, as in fprintf. gzprintf returns the number of uncompressed bytes actually written (0 in case of error).
*int gzputs (gzFile file, const char *s);*
Writes the given null-terminated string to the compressed file, excluding the terminating null character.
gzputs returns the number of characters written, or -1 in case of error.

*char * gzgets (gzFile file, char *buf, int len);*
Reads bytes from the compressed file until len-1 characters are read, or a newline character is read and transferred to buf, or an end-of-file condition is encountered. The string is then terminated with a null character.
gzgets returns buf, or Z_NULL in case of error.

int gzputc (gzFile file, int c);
Writes c, converted to an unsigned char, into the compressed file. gzputc returns the value that was written, or -1 in case of error.
int gzgetc (gzFile file);
Reads one byte from the compressed file. gzgetc returns this byte or -1 in case of end of file or error.
int gzflush (gzFile file, int flush);
Flushes all pending output into the compressed file. The parameter flush is as in the deflate() function. The return value is the zlib error number (see function gzerror below). gzflush returns Z_OK if the flush parameter is Z_FINISH and all output could be flushed.
gzflush should be called only when strictly necessary because it can degrade compression.

z_off_t gzseek (gzFile file, z_off_t offset, int whence);
Sets the starting position for the next gzread or gzwrite on the given compressed file. The offset represents a number of bytes in the uncompressed data stream. The whence parameter is defined as in lseek(2); the value SEEK_END is not supported.
If the file is opened for reading, this function is emulated but can be extremely slow. If the file is opened for writing, only forward seeks are supported ; gzseek then compresses a sequence of zeroes up to the new starting position.
gzseek returns the resulting offset location as measured in bytes from the beginning of the uncompressed stream, or -1 in case of error, in particular if the file is opened for writing and the new starting position would be before the current position.

int gzrewind (gzFile file);
Rewinds the given file. This function is supported only for reading.
gzrewind(file) is equivalent to (int)gzseek(file, 0L, SEEK_SET)

z_off_t gztell (gzFile file);
Returns the starting position for the next gzread or gzwrite on the given compressed file. This position represents a number of bytes in the uncompressed data stream.
gztell(file) is equivalent to gzseek(file, 0L, SEEK_CUR)

int gzeof (gzFile file);
Returns 1 when EOF has previously been detected reading the given input stream, otherwise zero.
int gzclose (gzFile file);
Flushes all pending output if necessary, closes the compressed file and deallocates all the (de)compression state. The return value is the zlib error number (see function gzerror below).
*const char * gzerror (gzFile file, int *errnum);*
Returns the error message for the last error which occurred on the given compressed file. errnum is set to zlib error number. If an error occurred in the file system and not in the compression library, errnum is set to Z_ERRNO and the application may consult errno to get the exact error code.

基本函数:
函数列表:
*const char * zlibVersion (void);*
int deflateInit (z_streamp strm, int level);
int deflate (z_streamp strm, int flush);
int deflateEnd (z_streamp strm);
int inflateInit (z_streamp strm);
int inflate (z_streamp strm, int flush);
int inflateEnd (z_streamp strm);
函数说明：
*const char * zlibVersion (void);*
应用程序会比较*zlibVersion*和*ZLIB_VERSION*的一致性。
如果第一个字不同，说明*zlib*和应用程序使用的*zlib.h*是不一致的。这个检查将被*defalteInit*和*infalteInit*自动调用。
int deflateInit (z_streamp strm, int level);
为压缩初始化内部流的状态。
Initializes the internal stream state for compression. The fields zalloc, zfree and opaque must be initialized before by the caller. If zalloc and zfree are set to Z_NULL, deflateInit updates them to use default allocation functions.

The compression level must be Z_DEFAULT_COMPRESSION, or between 0 and 9: 1 gives best speed, 9 gives best compression, 0 gives no compression at all (the input data is simply copied a block at a time).

Z_DEFAULT_COMPRESSION requests a default compromise between speed and compression (currently equivalent to level 6).

deflateInit returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_STREAM_ERROR if level is not a valid compression level, Z_VERSION_ERROR if the zlib library version (zlib_version) is incompatible with the version assumed by the caller (ZLIB_VERSION). msg is set to null if there is no error message. deflateInit does not perform any compression: this will be done by deflate().

int deflate (z_streamp strm, int flush);
deflate compresses as much data as possible, and stops when the input buffer becomes empty or the output buffer becomes full. It may introduce some output latency (reading input without producing any output) except when forced to flush.
The detailed semantics are as follows. deflate performs one or both of the following actions:

Compress more input starting at next_in and update next_in and avail_in accordingly. If not all input can be processed (because there is not enough room in the output buffer), next_in and avail_in are updated and processing will resume at this point for the next call of deflate().
Provide more output starting at next_out and update next_out and avail_out accordingly. This action is forced if the parameter flush is non zero. Forcing flush frequently degrades the compression ratio, so this parameter should be set only when necessary (in interactive applications). Some output may be provided even if flush is not set.
Before the call of deflate(), the application should ensure that at least one of the actions is possible, by providing more input and/or consuming more output, and updating avail_in or avail_out accordingly ; avail_out should never be zero before the call. The application can consume the compressed output when it wants, for example when the output buffer is full (avail_out == 0), or after each call of deflate(). If deflate returns Z_OK and with zero avail_out, it must be called again after making room in the output buffer because there might be more output pending.

If the parameter flush is set to Z_SYNC_FLUSH, all pending output is flushed to the output buffer and the output is aligned on a byte boundary, so that the decompressor can get all input data available so far. (In particular avail_in is zero after the call if enough output space has been provided before the call.) Flushing may degrade compression for some compression algorithms and so it should be used only when necessary.

If flush is set to Z_FULL_FLUSH, all output is flushed as with Z_SYNC_FLUSH, and the compression state is reset so that decompression can restart from this point if previous compressed data has been damaged or if random access is desired. Using Z_FULL_FLUSH too often can seriously degrade the compression.

If deflate returns with avail_out == 0, this function must be called again with the same value of the flush parameter and more output space (updated avail_out), until the flush is complete (deflate returns with non-zero avail_out).

If the parameter flush is set to Z_FINISH, pending input is processed, pending output is flushed and deflate returns with Z_STREAM_END if there was enough output space ; if deflate returns with Z_OK, this function must be called again with Z_FINISH and more output space (updated avail_out) but no more input data, until it returns with Z_STREAM_END or an error. After deflate has returned Z_STREAM_END, the only possible operations on the stream are deflateReset or deflateEnd.

Z_FINISH can be used immediately after deflateInit if all the compression is to be done in a single step. In this case, avail_out must be at least 0.1% larger than avail_in plus 12 bytes. If deflate does not return Z_STREAM_END, then it must be called again as described above.

deflate() sets strm-> adler to the adler32 checksum of all input read so far (that is, total_in bytes).

deflate() may update data_type if it can make a good guess about the input data type (Z_ASCII or Z_BINARY). In doubt, the data is considered binary. This field is only for information purposes and does not affect the compression algorithm in any manner.

deflate() returns Z_OK if some progress has been made (more input processed or more output produced), Z_STREAM_END if all input has been consumed and all output has been produced (only when flush is set to Z_FINISH), Z_STREAM_ERROR if the stream state was inconsistent (for example if next_in or next_out was NULL), Z_BUF_ERROR if no progress is possible (for example avail_in or avail_out was zero).

int deflateEnd (z_streamp strm);
All dynamically allocated data structures for this stream are freed. This function discards any unprocessed input and does not flush any pending output.
deflateEnd returns Z_OK if success, Z_STREAM_ERROR if the stream state was inconsistent, Z_DATA_ERROR if the stream was freed prematurely (some input or output was discarded). In the error case, msg may be set but then points to a static string (which must not be deallocated).

int inflateInit (z_streamp strm);
Initializes the internal stream state for decompression. The fields next_in, avail_in, zalloc, zfree and opaque must be initialized before by the caller. If next_in is not Z_NULL and avail_in is large enough (the exact value depends on the compression method), inflateInit determines the compression method from the zlib header and allocates all data structures accordingly ; otherwise the allocation will be deferred to the first call of inflate. If zalloc and zfree are set to Z_NULL, inflateInit updates them to use default allocation functions.
inflateInit returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_VERSION_ERROR if the zlib library version is incompatible with the version assumed by the caller. msg is set to null if there is no error message. inflateInit does not perform any decompression apart from reading the zlib header if present: this will be done by inflate(). (So next_in and avail_in may be modified, but next_out and avail_out are unchanged.)

int inflate (z_streamp strm, int flush);
inflate decompresses as much data as possible, and stops when the input buffer becomes empty or the output buffer becomes full. It may some introduce some output latency (reading input without producing any output) except when forced to flush.
The detailed semantics are as follows. inflate performs one or both of the following actions:

Decompress more input starting at next_in and update next_in and avail_in accordingly. If not all input can be processed (because there is not enough room in the output buffer), next_in is updated and processing will resume at this point for the next call of inflate().
Provide more output starting at next_out and update next_out and avail_out accordingly. inflate() provides as much output as possible, until there is no more input data or no more space in the output buffer (see below about the flush parameter).
Before the call of inflate(), the application should ensure that at least one of the actions is possible, by providing more input and/or consuming more output, and updating the next_* and avail_* values accordingly. The application can consume the uncompressed output when it wants, for example when the output buffer is full (avail_out == 0), or after each call of inflate(). If inflate returns Z_OK and with zero avail_out, it must be called again after making room in the output buffer because there might be more output pending.

If the parameter flush is set to Z_SYNC_FLUSH, inflate flushes as much output as possible to the output buffer. The flushing behavior of inflate is not specified for values of the flush parameter other than Z_SYNC_FLUSH and Z_FINISH, but the current implementation actually flushes as much output as possible anyway.

inflate() should normally be called until it returns Z_STREAM_END or an error. However if all decompression is to be performed in a single step (a single call of inflate), the parameter flush should be set to Z_FINISH. In this case all pending input is processed and all pending output is flushed ; avail_out must be large enough to hold all the uncompressed data. (The size of the uncompressed data may have been saved by the compressor for this purpose.) The next operation on this stream must be inflateEnd to deallocate the decompression state. The use of Z_FINISH is never required, but can be used to inform inflate that a faster routine may be used for the single inflate() call.

If a preset dictionary is needed at this point (see inflateSetDictionary below), inflate sets strm-adler to the adler32 checksum of the dictionary chosen by the compressor and returns Z_NEED_DICT ; otherwise it sets strm-> adler to the adler32 checksum of all output produced so far (that is, total_out bytes) and returns Z_OK, Z_STREAM_END or an error code as described below. At the end of the stream, inflate() checks that its computed adler32 checksum is equal to that saved by the compressor and returns Z_STREAM_END only if the checksum is correct.

inflate() returns Z_OK if some progress has been made (more input processed or more output produced), Z_STREAM_END if the end of the compressed data has been reached and all uncompressed output has been produced, Z_NEED_DICT if a preset dictionary is needed at this point, Z_DATA_ERROR if the input data was corrupted (input stream not conforming to the zlib format or incorrect adler32 checksum), Z_STREAM_ERROR if the stream structure was inconsistent (for example if next_in or next_out was NULL), Z_MEM_ERROR if there was not enough memory, Z_BUF_ERROR if no progress is possible or if there was not enough room in the output buffer when Z_FINISH is used. In the Z_DATA_ERROR case, the application may then call inflateSync to look for a good compression block.

int inflateEnd (z_streamp strm);
所有为这个*stream*动态分派的数据结构在这被释放。 这个函数丢弃所有未处理的输入和不输出任何未决的输出。
如果成功， *inflateEnd* 返回 *Z_OK* ； 如果*stream*是不一致的 返回*Z_STREAM_ERROR*，
在错误情形中，*msg*信息可能被设置，然后指向一个静态字符串。

高级函数:
以下函数应用于特殊应用程序:
函数列表:

int deflateInit2 (z_streamp strm,
*int deflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
int deflateCopy (z_streamp dest, z_streamp source);
int deflateReset (z_streamp strm);

int deflateParams (z_streamp strm, int level, int strategy);
int inflateInit2 (z_streamp strm, int windowBits);
*int inflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
int inflateSync (z_streamp strm);
int inflateReset (z_streamp strm);
函数说明：

int deflateInit2 (z_streamp strm, int level, int method, int windowBits, int memLevel, int strategy);

This is another version of deflateInit with more compression options. The fields next_in, zalloc, zfree and opaque must be initialized before by the caller.
The method parameter is the compression method. It must be Z_DEFLATED in this version of the library.

The windowBits parameter is the base two logarithm of the window size (the size of the history buffer). It should be in the range 8..15 for this version of the library. Larger values of this parameter result in better compression at the expense of memory usage. The default value is 15 if deflateInit is used instead.

The memLevel parameter specifies how much memory should be allocated for the internal compression state. memLevel=1 uses minimum memory but is slow and reduces compression ratio ; memLevel=9 uses maximum memory for optimal speed. The default value is 8. See zconf.h for total memory usage as a function of windowBits and memLevel.

The strategy parameter is used to tune the compression algorithm. Use the value Z_DEFAULT_STRATEGY for normal data, Z_FILTERED for data produced by a filter (or predictor), or Z_HUFFMAN_ONLY to force Huffman encoding only (no string match). Filtered data consists mostly of small values with a somewhat random distribution. In this case, the compression algorithm is tuned to compress them better. The effect of Z_FILTERED is to force more Huffman coding and less string matching ; it is somewhat intermediate between Z_DEFAULT and Z_HUFFMAN_ONLY. The strategy parameter only affects the compression ratio but not the correctness of the compressed output even if it is not set appropriately.

deflateInit2 returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_STREAM_ERROR if a parameter is invalid (such as an invalid method). msg is set to null if there is no error message. deflateInit2 does not perform any compression: this will be done by deflate().

*int deflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
Initializes the compression dictionary from the given byte sequence without producing any compressed output. This function must be called immediately after deflateInit, deflateInit2 or deflateReset, before any call of deflate. The compressor and decompressor must use exactly the same dictionary (see inflateSetDictionary).
The dictionary should consist of strings (byte sequences) that are likely to be encountered later in the data to be compressed, with the most commonly used strings preferably put towards the end of the dictionary. Using a dictionary is most useful when the data to be compressed is short and can be predicted with good accuracy ; the data can then be compressed better than with the default empty dictionary.

Depending on the size of the compression data structures selected by deflateInit or deflateInit2, a part of the dictionary may in effect be discarded, for example if the dictionary is larger than the window size in deflate or deflate2. Thus the strings most likely to be useful should be put at the end of the dictionary, not at the front.

Upon return of this function, strm-> adler is set to the Adler32 value of the dictionary ; the decompressor may later use this value to determine which dictionary has been used by the compressor. (The Adler32 value applies to the whole dictionary even if only a subset of the dictionary is actually used by the compressor.)

deflateSetDictionary returns Z_OK if success, or Z_STREAM_ERROR if a parameter is invalid (such as NULL dictionary) or the stream state is inconsistent (for example if deflate has already been called for this stream or if the compression method is bsort). deflateSetDictionary does not perform any compression: this will be done by deflate().

int deflateCopy (z_streamp dest, z_streamp source);
Sets the destination stream as a complete copy of the source stream.
This function can be useful when several compression strategies will be tried, for example when there are several ways of pre-processing the input data with a filter. The streams that will be discarded should then be freed by calling deflateEnd. Note that deflateCopy duplicates the internal compression state which can be quite large, so this strategy is slow and can consume lots of memory.

deflateCopy returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_STREAM_ERROR if the source stream state was inconsistent (such as zalloc being NULL). msg is left unchanged in both source and destination.

int deflateReset (z_streamp strm);
This function is equivalent to deflateEnd followed by deflateInit, but does not free and reallocate all the internal compression state. The stream will keep the same compression level and any other attributes that may have been set by deflateInit2.
deflateReset returns Z_OK if success, or Z_STREAM_ERROR if the source stream state was inconsistent (such as zalloc or state being NULL).

int deflateParams (z_streamp strm, int level, int strategy);
Dynamically update the compression level and compression strategy. The interpretation of level and strategy is as in deflateInit2. This can be used to switch between compression and straight copy of the input data, or to switch to a different kind of input data requiring a different strategy. If the compression level is changed, the input available so far is compressed with the old level (and may be flushed); the new level will take effect only at the next call of deflate().
Before the call of deflateParams, the stream state must be set as for a call of deflate(), since the currently available input may have to be compressed and flushed. In particular, strm-> avail_out must be non-zero.

deflateParams returns Z_OK if success, Z_STREAM_ERROR if the source stream state was inconsistent or if a parameter was invalid, Z_BUF_ERROR if strm->avail_out was zero.

int inflateInit2 (z_streamp strm, int windowBits);
This is another version of inflateInit with an extra parameter. The fields next_in, avail_in, zalloc, zfree and opaque must be initialized before by the caller.
The windowBits parameter is the base two logarithm of the maximum window size (the size of the history buffer). It should be in the range 8..15 for this version of the library. The default value is 15 if inflateInit is used instead. If a compressed stream with a larger window size is given as input, inflate() will return with the error code Z_DATA_ERROR instead of trying to allocate a larger window.

inflateInit2 returns Z_OK if success, Z_MEM_ERROR if there was not enough memory, Z_STREAM_ERROR if a parameter is invalid (such as a negative memLevel). msg is set to null if there is no error message. inflateInit2 does not perform any decompression apart from reading the zlib header if present: this will be done by inflate(). (So next_in and avail_in may be modified, but next_out and avail_out are unchanged.)

*int inflateSetDictionary (z_streamp strm, const Bytef *dictionary, uInt dictLength);*
Initializes the decompression dictionary from the given uncompressed byte sequence. This function must be called immediately after a call of inflate if this call returned Z_NEED_DICT. The dictionary chosen by the compressor can be determined from the Adler32 value returned by this call of inflate. The compressor and decompressor must use exactly the same dictionary (see deflateSetDictionary).
inflateSetDictionary returns Z_OK if success, Z_STREAM_ERROR if a parameter is invalid (such as NULL dictionary) or the stream state is inconsistent, Z_DATA_ERROR if the given dictionary doesn't match the expected one (incorrect Adler32 value). inflateSetDictionary does not perform any decompression: this will be done by subsequent calls of inflate().

int inflateSync (z_streamp strm);
Skips invalid compressed data until a full flush point (see above the description of deflate with Z_FULL_FLUSH) can be found, or until all available input is skipped. No output is provided.
inflateSync returns Z_OK if a full flush point has been found, Z_BUF_ERROR if no more input was provided, Z_DATA_ERROR if no flush point has been found, or Z_STREAM_ERROR if the stream structure was inconsistent. In the success case, the application may save the current current value of total_in which indicates where valid compressed data was found. In the error case, the application may repeatedly call inflateSync, providing more input each time, until success or end of the input data.

int inflateReset (z_streamp strm);
这个函数伴随*inflateInit*,跟*inflateEnd*是等价的,但不释放和在分配所有的内部解压缩状态。
这个*stream*保持被*inflateInit2*设置的属性。
如果成功， *inflateReset* 返回 *Z_OK* ； 如果*stream*是不一致的 返回*Z_STREAM_ERROR*，

校验函数
这些函数和压缩是没有关系的.但是被公开是因为他们在程序使用压缩库时,可能是有用的。
函数列表:

uLong adler32 (uLong adler, const Bytef *buf, uInt len);
uLong crc32 (uLong crc, const Bytef *buf, uInt len);
函数说明：

uLong adler32 (uLong adler, const Bytef *buf, uInt len);

Update a running Adler-32 checksum with the bytes buf[0..len-1] and return the updated checksum. If buf is NULL, this function returns the required initial value for the checksum.
An Adler-32 checksum is almost as reliable as a CRC32 but can be computed much faster. Usage example:

```
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);
Update a running crc with the bytes buf[0..len-1] and return the updated crc. If buf is NULL, this function returns the required initial value for the crc. Pre- and post-conditioning (one's complement) is performed within this function so it shouldn't be done by the application. Usage example:

```
uLong crc = crc32(0L, Z_NULL, 0);
while (read_buffer(buffer, length) != EOF) {
  crc = crc32(crc, buffer, length);
}
if (crc != original_crc) error();
```

---

struct z_stream_s
typedef struct z_stream_s {
  Bytef   *next_in;
  uInt   avail_in;
  uLong   total_in;
  Bytef   *next_out;
  uInt   avail_out;
  uLong   total_out;
  char   *msg;
  struct internal_state FAR *state;
  alloc_func zalloc;
  free_func zfree;
  voidpf   opaque;
  int   data_type;
  uLong   adler;
  uLong   reserved;
} z_stream ;
typedef z_stream FAR * z_streamp; ÿ

The application must update next_in and avail_in when avail_in has dropped to zero. It must update next_out and avail_out when avail_out has dropped to zero. The application must initialize zalloc, zfree and opaque before calling the init function. All other fields are set by the compression library and must not be updated by the application.
The opaque value provided by the application will be passed as the first parameter for calls of zalloc and zfree. This can be useful for custom memory management. The compression library attaches no meaning to the opaque value.
zalloc must return Z_NULL if there is not enough memory for the object. If zlib is used in a multi-threaded application, zalloc and zfree must be thread safe.
On 16-bit systems, the functions zalloc and zfree must be able to allocate exactly 65536 bytes, but will not be required to allocate more than this if the symbol MAXSEG_64K is defined (see zconf.h). WARNING: On MSDOS, pointers returned by zalloc for objects of exactly 65536 bytes *must* have their offset normalized to zero. The default allocation function provided by this library ensures this (see zutil.c). To reduce memory requirements and avoid any allocation of 64K objects, at the expense of compression ratio, compile the library with -DMAX_WBITS=14 (see zconf.h).
The fields total_in and total_out can be used for statistics or progress reports. After compression, total_in holds the total size of the uncompressed data and may be saved for use in the decompressor (particularly if the decompressor wants to decompress everything in a single step).

常量:

```
#define Z_NO_FLUSH      0
#define Z_PARTIAL_FLUSH 1

#define Z_SYNC_FLUSH    2
#define Z_FULL_FLUSH    3
#define Z_FINISH        4
#define Z_OK            0
#define Z_STREAM_END    1
#define Z_NEED_DICT     2
#define Z_ERRNO        (-1)
#define Z_STREAM_ERROR (-2)
#define Z_DATA_ERROR   (-3)
#define Z_MEM_ERROR    (-4)
#define Z_BUF_ERROR    (-5)
#define Z_VERSION_ERROR (-6)
#define Z_NO_COMPRESSION         0
#define Z_BEST_SPEED             1
#define Z_BEST_COMPRESSION       9
#define Z_DEFAULT_COMPRESSION  (-1)
#define Z_FILTERED               1
#define Z_HUFFMAN_ONLY           2
#define Z_DEFAULT_STRATEGY       0
#define Z_BINARY   0
#define Z_ASCII    1
#define Z_UNKNOWN  2
#define Z_DEFLATED   8
#define Z_NULL  0
#define zlib_version zlibVersion()
```

Misc
deflateInit 和 `inflateInit` 是检查`zlib`版本和`z_stream`的编译器`view`的宏.
另外一些函数:

```
const char * zError (int err);
int inflateSyncPoint (z_streamp z);
const uLongf * get_crc_table (void);
```

xiaoyeyopulei
码龄13年 暂无认证
44 原创 | 19万+ 周排名 | 83万+ 总排名 | 18万+ 访问 | 等级
2325 积分 | 56 粉丝 | 38 获赞 | 29 评论 | 130 收藏
私信 关注

热门文章

Linux多线程学习（三）pthread_key_create 22394

Linux多线程学习（七）sched_yield 14759

Linux多线程学习（八）pthread_setschedparam 13259

OpenSSL 做3DES加密 实现 9767

中文分词——正向最大匹配法 9547

最新评论

Linux多线程学习（七）sched_yield
Mo先生: 代码段有点乱哦

Linux多线程学习（三）pthread_key_cre…
清歌美酒: 请问一下，如果使用的是线程池，线程不退出的情况下，如何触发析构函…

Linux多线程学习（三）pthread_key_cre…
tyrocjl: 谢谢博主，解答了我的疑惑。 我的编译器版本为gcc (Ubuntu 7.5.0-3ubuntu…

Linux多线程学习（七）sched_yield
中华田园巨龙: 。。。。。。你这怎么还有一段没排版

遗传算法
扉首HYQ: java语言

您愿意向朋友推荐“博客详情页”吗？

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

红黑树实现

蚁群算法

中文分词选取-依概率选取

2014年 9篇 2013年 5篇
2012年 43篇

openwrt-17.01.6.tar.gz dl1 01-18
openwrt-17.01.6.tar.gz 下载dl 包 一共两个这是第一个 dl1.tar.gz

zlib 封装简单接口使用 12-23
实际应用中有时候会遇到需要处理 ZIP 压缩解压的情况，这时候我们有大概三种选择： 调用 rar.exe, unzip.exe 等 使用某现成库 完全手写 第一种虽然能完…

抢沙发
优质评论可以帮助作者获得更高权重
评论

zlib库剖析(一)_damenhanter的专栏_gzwrite 7-17
gzguts.h和gzlib.c:读写gzip文件的通用实现,包括gzopen(), gzdopen(), gzbuffer(), gzrewind(), gzseek(), gztell(), gzoffset(), gzeof(), gzerror(), gzclearerr()…

zlib压缩解压库 - CSDN博客 12-13
gzFile gzdopen (int fd, const char *mode); int gzsetparams (gzFile file, int level, int strategy); int gzread (gzFile file, voidp buf, unsigned len); int gzwrite …

详解 zlib 函数库 weixin_34289454的博客 399
2019独角兽企业重金招聘Python工程师标准>>> …

使用TStream*的gzip文件格式的压缩和解压函数 C++Builder加油站 2761
在论坛里回答一个关于在内存中直接对gzip格式解压的帖子时试验出来的代码帖子:http://topic.csdn.net/u/20071015/16/14b00c8e-767d-4608-966a-0b5d…

zlib库compress和uncompress函数的使用方法 热门推荐 图灵狗的专栏 6万+
zlib(http://zlib.net/)提供了简洁高效的In-Memory数据压缩和解压缩系列API函数，很多应用都会用到这个库，其中compress和uncompress函数是最基本也…

zlib库剖析(1)：实现概览 Jack Zhou的专栏 1万+
本文整理自zlib.net以及zlib 1.2.7的手册页http://zlib.net/manual.html。 zlib是一套免费、通用、法律上不受限制的无损数据压缩库，可以在任何硬件及操…

zlib使用 学习笔记 5894
zlib.net也被墙了，晕 Zlib函数列表 实用函数 int compress (Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen); int compress2 (Bytef *

DevCpp中Zlib包的使用 落叶小灌木.BLOG 2017
英文版本:http://www.gzip.org/zlib/manual.html#compress"zlib" 通用压缩程序库 版本1.1.4,2002年3月11日版本所有?(C) 1995-2002 Jean-loup Gailly and …

zlib使用gzopen("test.gz","rb")打开gz文件，程序崩溃 kanguolaikanguolaik的专栏 3263
zlib使用gzopen("test.gz","rb")打开gz文件，程序崩溃。 编译环境：VS2010. zlib版本：1.2.8 代码如下： int main(int argc, char *argv[]) { gzFile gzfp = gz…

zlib通用的压缩库简介 2152
zlib 是通用的压缩库，提供了一套 in-memory 压缩和解压函数，并能检测解压出来的数据的完整性(integrity)。zlib 也支持读写 gzip (.gz) 格式的文件。下面…

libxm2使用未定义的引用gzdopen lzma_code lzma_properties_decode lzma_end 发呆的程序猿 420
./lib/libxml2.a(xzlib.o)：在函数'xz_decomp'中： /home/shenxuebing/桌面/xml/libxml2-2.9.9/xzlib.c:581：对'lzma_code'未定义的引用 ./lib/libxml2.a(xzlib.o)…

用文件句柄读取内存内容 yanjing2407的专栏 917
下面的问题和本文很可能是一个意思: How to read memory using file handler?How to read string as file? 有的C语言API只接受文件句柄作为参数。这时我…

语言写printf不输出_你不知道c语言写的程序要多块——以NGS fastq文件reads计数为例... 最新发布 weixin_31881743的博客 23
本人 以fastq.gz文件计数为例分别以perl语言和c语言实现了代码，具体如下:#!/usr/bin/perl -w## Copyright (c) xuxiong 2020# Writer: xuxiong # Program …