Erlang disk_log

xushiwei@gmail.com

2007-11-26

Summary

- logger
- disk_log
  - halt logs / wrap logs
  - internal / external format
  - owner / user process
  - sync / async
  - distributed / local disk log
- appendix

Who need logger?

- Database
- Erlana
- Spider
- Most distributed servers ...

Why need logger?

- Why I am interested in logger?
  - Robust log (cannot shutdown or lose data)
  - Distributed log (huge data)
  - Load balance (no read/write bottleneck)

logger - disk_log

- Is disk_log fit for us?

disk_log

- disk_log is a disk based term logger which makes it possible to efficiently log items on files.
- For the sake of efficiency, items are always written to files as binaries.

halt logs / wrap logs

- halt logs
  - A halt log appends items to a single file, the size of which may or may not be limited by the disk_log module.
- **wrap logs**
  - A wrap log **utilizes a sequence of wrap log files of limited size**.
  - As a wrap log file has been filled up, further items are logged onto to the next file in the sequence, starting all over with the first file when the last file has been filled up.

internal / external format

- **internal format**
  - Supports **automatic repair** of log files that have not been properly closed.
  - **Efficiently read** logged items in **chunks** using a set of functions defined in this module. In fact, this is the only way to read internally formatted logs.
  - An item logged to an internally formatted log must not occupy more than 4 GB of disk space (the size must fit in 4 bytes).
- external format
  - **Up to the user** to read the logged deep byte lists.
  - Cannot automatic repair.

cont.

- internal format
  - When using the internal format for logs, the functions **log/2, log_terms/2, alog/2, and alog_terms/2** should be used. These functions log one or more Erlang terms.
- external format
  - By prefixing each of the functions with a *b* (for "binary") we get the corresponding blog functions for the external format.
  - These functions log one or more deep lists of bytes or, alternatively, binaries of deep lists of bytes.
    - eg. log the string "hello" in ASCII format, we can use disk_log:blog(Log, "hello"), or disk_log:blog(Log, list_to_binary("hello")).

sync / async

- sync
  - Items can be logged synchronously by using the functions log/2, blog/2, log_terms/2 and blog_terms/2.
    - For each of these functions, the caller is put on hold until the items have been logged (but not necessarily written, use sync/1 to ensure that).
- async
  - Use functions **alog/2, balog/2, alog_terms/2, balog_terms/2**, by prefixing with a *a* (for "async").
  - Asynchronous functions do not wait for the disk log process to actually write the items to the file, but return the control to the caller more or less immediately.

cont.

- Error report
  - When used synchronously the disk log module replies with an error message.
  - When called asynchronously, the disk log module does not know where to send the error message. Instead owners subscribing to notifications will receive an **error_status** message.

owner / user process

- A process that opens a disk log can either be an owner or an anonymous user of the disk log.
- owner process
  - Each owner is linked to the disk log process, and the disk log is closed by the owner should the owner terminate.
  - Owners can subscribe to notifications, messages of the form {disk_log, Node, Log, Info} that are sent from the disk log process when certain events occur, see the commands below and in particular the open/1 option notify.
  - There can be several owners of a log, but a process cannot own a log more than once.

cont.

- user process
  - For a disk log process to properly close its file and terminate, **it must be closed by its owners** and once by some non-owner process for each time the log was used anonymously; the users are counted, and there must not be any users left when the disk log process terminates.
  - One process may open the log as a user more than once.

distributed / local disk log

- distributed disk log
  - A collection of open disk logs with the same name running on different nodes. If requests made to any one of the logs are automatically made to the other logs as well.
  - The members of such a collection will be called individual distributed disk logs, or just distributed disk logs if there is no risk of confusion. There is no order between the members of such a collection.

appendix

- open / reopen
- close / lclose
- truncate

- alog / log
- sync
- inc_wrap_file

- chunk / chunk_step
- chunk_info

- block / unblock

- info
- pid2name
- accessible_logs
- change_header / change_notify / change_size

handle operations

- open
- reopen
- close
- lclose
- truncate

open

- *open([Opt])*
  -> {ok, Log} | {error, Reason} |
  {repaired, Log, {recovered, Rec},
  {badbytes, Bad}}
  - *Opt =*
    {name, term()} | {file, FileName} |
    {linkto, LinkTo} |
    {repair, Repair} |
    {type, Type} |
    {format, Format} |
    {size, Size} |
    {distributed, [Node]} |
    {notify, bool()} |
    {head, Head} | {head_func, {M,F,A}} |
    {mode, Mode}

cont.

- *{name, Log}* specifies the name of the log. This is the name which must be passed on as a parameter in all subsequent logging operations. A name must always be supplied.
- *{file, FileName}* specifies the name of the file which will be used for logged terms. If this value is omitted and the name of the log is either an atom or a string, the file name will default to lists:concat([Log, ".LOG"]) for halt logs. For wrap logs, this will be the base name of the files. Each file in a wrap log will be called <base_name>.N, where N is an integer. Each wrap log will also have two files called <base_name>.idx and <base_name>.siz.
- *{linkto, LinkTo}*. If LinkTo is a pid, that pid becomes an owner of the log. If LinkTo is none the log records that it is used anonymously by some process by incrementing the users counter. By default, the process which calls open/1 owns the log.
- *{repair, Repair}*. If Repair is true, the current log file will be repaired, if needed. As the restoration is initiated, a message is output on the error log. If false is given, no automatic repair will be attempted. Instead, the tuple {error, {need_repair, Log}} is returned if an attempt is made to open a corrupt log file. If truncate is given, the log file will be truncated, creating an empty log. Default is true, which has no effect on logs opened in read-only mode.

cont.

- *{type, Type}* is the type of the log. Default is halt.
- *{format, Format}* specifies the format of the disk log. Default is internal.
- *{size, Size}* specifies the size of the log. When a halt log has reached its maximum size, all attempts to log more items are rejected. The default size is infinity, which for halt implies that there is no maximum size. For wrap logs, the Size parameter may be either a pair *{MaxNoBytes, MaxNoFiles}* or infinity. In the latter case, if the files of an already existing wrap log with the same name can be found, the size is read from the existing wrap log, otherwise an error is returned. Wrap logs write at most MaxNoBytes bytes on each file and use MaxNoFiles files before starting all over with the first wrap log file. Regardless of MaxNoBytes, at least the header (if there is one) and one item is written on each wrap log file before wrapping to the next file. When opening an existing wrap log, it is not necessary to supply a value for the option Size, but any supplied value must equal the current size of the log, otherwise the tuple *{error, {size_mismatch, CurrentSize, NewSize}}* is returned.
- *{distributed, Nodes}*. This option can be used for adding members to a distributed disk log. The default value is [], which means that the log is local on the current node.

cont.

- *{notify, bool()}.* If true, the owners of the log are notified when certain events occur in the log. Default is false. The owners are sent one of the following messages when an event occurs:
  - {disk_log, Node, Log, {wrap, NoLostItems}} is sent when a wrap log has filled up one of its files and a new file is opened. NoLostItems is the number of previously logged items that have been lost when truncating existing files.
  - {disk_log, Node, Log, {truncated, NoLostItems}} is sent when a log has been truncated or reopened. For halt logs NoLostItems is the number of items written on the log since the disk log process was created. For wrap logs NoLostItems is the number of items on all wrap log files.
  - {disk_log, Node, Log, {read_only, Items}} is sent when an asynchronous log attempt is made to a log file opened in read-only mode. Items is the items from the log attempt.
  - {disk_log, Node, Log, {blocked_log, Items}} is sent when an asynchronous log attempt is made to a blocked log that does not queue log attempts. Items is the items from the log attempt.
  - {disk_log, Node, Log, {format_external, Items}} is sent when alog/2 or alog_terms/2 is used for internally formatted logs. Items is the items from the log attempt.
  - {disk_log, Node, Log, full} is sent when an attempt to log items to a wrap log would write more bytes than the limit set by the size option.
  - {disk_log, Node, Log, {error_status, Status}} is sent when the error status changes. The error status is defined by the outcome of the last attempt to log items to a the log or to truncate the log or the last use of sync/1, inc_wrap_file/1 or change_size/2. Status is one of ok and {error, Error}, the former being the initial value.

cont.

- *{head, Head}* specifies a header to be written first on the log file. If the log is a wrap log, the item Head is written first in each new file. Head should be a term if the format is internal, and a deep list of bytes (or a binary) otherwise. Default is none, which means that no header is written first on the file.
- *{head_func, {M,F,A}}* specifies a function to be called each time a new log file is opened. The call M:F(A) is assumed to return {ok, Head}. The item Head is written first in each file. Head should be a term if the format is internal, and a deep list of bytes (or a binary) otherwise.
- *{mode, Mode}* specifies if the log is to be opened in read-only or read-write mode. It defaults to read_write.

reopen

- *reopen(Log, File)*
- *reopen(Log, File, Head)*
- *breopen(Log, File, BHead)*
  -> ok | {error, Reason}

  – Rename the log file to File and then re-create a new log file. In case of a wrap log, File is used as the base name of the renamed files.
  – By default the header given to open/1 is written first in the newly opened log file, but if the Head or the BHead argument is given, this item is used instead. The header argument is used once only; next time a wrap log file is opened, the header given to open/1 is used.

close

- *close(Log) -> ok | {error, Reason}*
  - Closes a local or distributed disk log properly.
  - An internally formatted log must be closed before the Erlang system is stopped, otherwise the log is regarded as unclosed and the automatic repair procedure will be activated next time the log is opened.
  - The disk log process in not terminated as long as there are owners or users of the log. It should be stressed that **each and every owner must close the log**, possibly by terminating, and that any other process - not only the processes that have opened the log anonymously - can decrement the users counter by closing the log. Attempts to close a log by a process that is not an owner are simply ignored if there are no users.
  - If the log is blocked by the closing process, the log is also unblocked.

lclose

- *lclose(Log)*
- *lclose(Log, Node) -> ok | {error, Reason}*

  – Closes a local log or an individual distributed log on the current node.

  – *lclose(Log)* is equivalent to *lclose(Log, node())*.

truncate

- *truncate(Log)*
- *truncate(Log, Head)*
- *btruncate(Log, BHead) -> ok | {error, Reason}*

  - Remove all items from a disk log.
  - If the Head or the BHead argument is given, this item is written first in the newly truncated log, otherwise the header given to open/1 is used. The header argument is only used once; next time a wrap log file is opened, the header given to open/1 is used.

write operations

- alog
- log
- sync
- inc_wrap_file

alog

- *alog(Log, Term),*
- *balog(Log, Bytes)*

  -> ok | {error, Reason}

  – Asynchronously append an item to a disk log.
  – The function *alog(Log, Term)* equals *balog(Log, term_to_binary(Term))*, and **is used for internally formatted logs** only.
  – *Reason* can be *no_such_log*, if *Log* is invalid.

cont.

- *alog_terms(Log, [Term]),*
- *balog_terms(Log, [Bytes])*

  -> ok | {error, Reason}

  – Asynchronously append a list of items to a disk log.

log

- *log(Log, Term)*
- *blog(Log, Bytes) -> ok | {error, Reason}*

  - Synchronously append a term to a disk log. They return ok or *{error, Reason}* when the term has been written to disk.
  - If the log is distributed, ok is always returned, unless all nodes are down.
  - Terms are written by means of the ordinary write() function of the operating system. Hence, there is no guarantee that the term has actually been written to the disk, it might linger in the operating system kernel for a while. To make sure the item is actually written to disk, the sync/1 function must be called.

cont.

- *log_terms(Log, TermList)*
- *blog_terms(Log, BytesList)*

  -> ok | {error, Reason}

  – Synchronously append a list of items to the log.

sync

- *sync(Log) -> ok | {error, Reason}*
  - Ensures that the contents of the log are actually written to the disk.
  - This is usually a rather expensive operation.

inc_wrap_file

- *inc_wrap_file(Log) -> ok | {error, Reason}*

  – Forces the internally formatted disk log to start logging to the next log file.
  – It can be used, for instance, in conjunction with *change_size/2* to reduce the amount of disk space allocated by the disk log.

read operations

- chunk
- chunk_step
- chunk_info

chunk

- *chunk(Log, Continuation, N = infinity)*
  -> {Continuation2, Terms} |
  {Continuation2, Terms, Badbytes} | eof | {error, Reason}
- *bchunk(Log, Continuation, N = infinity)*
  -> {Continuation2, Binaries} |
  {Continuation2, Binaries, Badbytes} | eof | {error, Reason}

  – Efficiently read the terms which have been appended to an internally formatted log. It minimizes disk I/O by reading 64 kilobyte chunks from the file.
  – The *bchunk* functions return the binaries read from the file; they do not call *binary_to_term*.
  – The first time *chunk* (or *bchunk*) is called, an initial continuation, the atom ***start***, must be provided.
  – *N* controls the maximum number of terms that are read from the log in each chunk. Default is *infinity*, which means that all the terms contained in the 64 kilobyte chunk are read. **If less than N terms are returned, this does not necessarily mean that the end of the file has been reached**.

chunk_step

- *chunk_step(Log, Continuation, Step)*
  -> {ok, Continuation2} | {error, Reason}

  - Search through an internally formatted wrap log.
  - It takes as argument a continuation as returned by *chunk/2,3, bchunk/2,3, or chunk_step/3*, and steps forward (or backward) step files in the wrap log.
  - If the atom ***start*** is given as continuation, a disk log to read terms from is chosen. A local or distributed disk log on the current node is preferred to an individual distributed log on some other node.
  - If the wrap log is not full because all files have not been used yet, *{error, end_of_log}* is returned if trying to step outside the log.

chunk_info

- *chunk_info(Continuation)*

  -> InfoList | {error, Reason}

  – Returns the following pair describing the chunk continuation returned by *chunk/2,3, bchunk/2,3, or chunk_step/3*:

    - *{node, Node}.* Terms are read from the disk log running on Node.

sync operations

- block
- unblock

block

- *block(Log, QueueLogRecords = true)*
  -> ok | {error, Reason}

  – Block a log.
  – If the blocking process is not an owner of the log, a temporary link is created between the disk log process and the blocking process. The link is used to ensure that the disk log is unblocked should the blocking process terminate without first closing or unblocking the log.
  – The blocking process can also use the functions *chunk/2,3, bchunk/2,3, chunk_step/3, and unblock/1* without being affected by the block. Any other attempt than those hitherto mentioned to **update or read a blocked log suspends the calling process until the log is unblocked or returns an error message** {blocked_log, Log}.

unblock

- unblock(Log) -> ok | {error, Reason}
  - Unblocks a log.
  - A log can only be unblocked by the blocking process.

meta operations

- info
- pid2name
- accessible_logs
- change_header / change_notify / change_size

info

- *info(Log) -> InfoList | {error, no_such_log}*
  - Returns a list of *{Tag, Value}* pairs describing the log.
  - If there is a disk log process running on the current node, that log is used as source of information, otherwise an individual distributed log on some other node is chosen.

cont.

- *{name, Log},* The name of the log.
- *{type, Type},* The type of the log as given by the open/1 option type.
- *{format, Format},* The format of the log as given by the open/1 option format.
- *{size, Size},* The size of the log as given by the open/1 option size.
- *{mode, Mode},* The mode of the log as given by the open/1 option mode.
- *{owners, [{pid(), Notify}]}* *Notify* is the value set by the open/1 option notify or the function change_notify/3 for the owners of the log.
- *{users, Users}* The number of anonymous users of the log, see the open/1 option linkto.
- *{status, Status}*, where Status is ok or {blocked, QueueLogRecords} as set by the functions block/1,2 and unblock/1.
- *{node, Node}*. The information returned by the current invocation of the info/1 function has been gathered from the disk log process running on Node.
- *{distributed, Dist}*. If the log is local on the current node, then Dist has the value local, otherwise all nodes where the log is distributed are returned as a list.

cont.

- for all logs opened in read_write mode
  - *{head, Head}.* Depending of the value of the open/1 options head and head_func or set by the function change_header/2, the value of Head is none (default), {head, H} (head option) or {M,F,A} (head_func option).
  - *{no_written_items, NoWrittenItems}*, where NoWrittenItems is the number of items written to the log since the disk log process was created.
- for halt logs opened in read_write mode
  - *{full, Full}*, where Full is true or false depending on whether the halt log is full or not.

cont.

- for wrap logs opened in read_write mode:
  - *{no_current_bytes, integer() >= 0}* The number of bytes written to the current wrap log file.
  - *{no_current_items, integer() >= 0}* The number of items written to the current wrap log file, header inclusive.
  - *{no_items, integer() >= 0}* The total number of items in all wrap log files.
  - *{current_file, integer()}* The ordinal for the current wrap log file in the range 1..MaxNoFiles, where MaxNoFiles is given by the open/1 option size or set by change_size/2.
  - *{no_overflows, {SinceLogWasOpened, SinceLastInfo}}*, where SinceLogWasOpened (SinceLastInfo) is the number of times a wrap log file has been filled up and a new one opened or inc_wrap_file/1 has been called since the disk log was last opened (info/1 was last called). The first time info/2 is called after a log was (re)opened or truncated, the two values are equal.

pid2name

- *pid2name(Pid) -> {ok, Log} | undefined*
  - Returns the name of the log given the pid of a disk log process on the current node, or *undefined* if the given pid is not a disk log process.
  - This function is meant to be used for debugging only.

accessible_logs

- *accessible_logs()*

  -> {[LocalLog], [DistributedLog]}

  – Returns **the names of the disk logs** accessible on the current node.

  – The first list contains local disk logs, and the second list contains distributed disk logs.

change_header

- *change_header(Log, Header)*

  -> ok | {error, Reason}

  - Changes the value of the head or head_func option of a disk log.
  - *Header = {head, Head} | {head_func, {M,F,A}}*

change_notify

- *change_notify(Log, Owner, Notify)*

  -> ok | {error, Reason}

  - Changes the value of the notify option for an owner of a disk log.
  - *Notify = true / false*

change_size

- *change_size(Log, Size)*
  -> ok | {error, Reason}

  – Changes the size of an open log.
  – *Size = int*
    - For a halt log it is always possible to increase the size, but it is not possible to decrease the size to something less than the current size of the file.
  – *Size = {MaxNoBytes, MaxNoFiles}*
    - For a wrap log it is always possible to increase both the size and number of files, as long as the number of files does not exceed 65000. If the maximum number of files is decreased, the change will not be valid until the current file is full and the log wraps to the next file. The redundant files will be removed next time the log wraps around, i.e. starts to log to file number 1.

cont.

- an example
  - assume that the old maximum number of files is 10 and that the new maximum number of files is 6.
    - If the current file number is not greater than the new maximum number of files, the files 7 to 10 will be removed when file number 6 is full and the log starts to write to file number 1 again.
    - Otherwise the files greater than the current file will be removed when the current file is full (e.g. if the current file is 8, the files 9 and 10); the files between new maximum number of files and the current file (i.e. files 7 and 8) will be removed next time file number 6 is full.

format_error

- *format_error(Error) -> Chars*
  - Format error message strings.

Thanks