**读取日志设计**

**Table of Contents**

* [1 功能划分](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-1)
  + [1.1 根据文件ID和文件偏移读取一批日志： ObLocatedLogReader](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-1-1)
  + [1.2 位置缓存和实际定位： ObDefaultLogLocator 的实现](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-1-2)
* [2 日志位置缓存： ObLogLocationCache](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-2)
* [3 实际定位： ObOnDiskLogLocator](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-3)
  + [3.1 由日志ID查询文件ID](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-3-1)
  + [3.2 由日志ID和文件ID查询文件偏移](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html#sec-3-2)

**1 功能划分**

ObPosLogReader 根据日志ID读取日志文件，返回给定ID(包括该ID)之后的一批日志。要完成读取功能，需要两个功能:

1. 根据日志ID快速定位日志文件ID和文件内偏移，这部分功能由 ObDefaultLogLocator 实现。
2. 根据日志文件ID和文件偏移读取日志的类，这部分功能由 ObLocatedLogReader 完成。

日志的位置信息定义如下:

struct ObLogLocation {

int64\_t log\_id\_;

int64\_t file\_id\_;

int64\_t offset\_;

};

ObDefaultLogLocator 和 ObLocatedLogReader 接口定义如下:

// get\_location()从日志ID获得location

// enter\_location()记录一个新的location,

// 只有在locator可以缓存location的情况下，enter\_location()才有意义；

// 否则，可以把enter\_location()定义为空。

class IObLogLocator {

// 当定位失败时 get\_location()返回 OB\_ENTRY\_NOT\_EXIST

virtual int get\_location(const int64\_t log\_id, ObLogLocation& location) = 0;

virtual int enter\_location(const ObLogLocation& location) = 0;

};

class IObLocatedLogReader {

virtual int read\_log(const int64\_t file\_id, const int64\_t offset, const int64\_t start\_log\_id, int64\_t& end\_log\_id,

char\* buf, const int64\_t len, int64\_t& read\_count) = 0;

};

ObPosLogReader 实现如下:

// IObLogSrc定义了获取日志的源的接口，recent\_log\_cache/pos\_log\_reader/fetched\_log\_buffer最后都要实现这个接口

class ObPosLogReader : public IObLogSrc

{

public:

virtual int get\_log(const int64\_t start\_log\_id, int64\_t& end\_log\_id,

char\* buf, const int64\_t len, int64\_t& read\_count);

private:

IObLogLocator\* log\_locator\_;

IObLocatedLogReader\* located\_log\_reader\_;

};

get\_log(start\_log\_id, end\_log\_id, buf, len, read\_count)

1. 使用 log\_locator\_.get\_location(log\_id, location) 得到文件ID和偏移
2. 使用 located\_log\_reader\_.read(location.file\_id\_, location.offset\_, start\_log\_id, end\_log\_id\_, buf, len, read\_count) 读取文件内容到 buf
3. 调用 log\_locator\_.enter\_location(end\_location) 缓存最后的日志位置。

实际使用时, 由 ObDefaultLogLocator 完成日志定位， ObLocatedLogReader 完成日志读取。

**1.1 根据文件ID和文件偏移读取一批日志： ObLocatedLogReader**

ObLocatedLogReader 目前实现为每次重新打开指定的文件，定位到给定的偏移，读取日志，读取的日志拷贝到给定的缓冲区内，

class ObLocatedLogReader : public IObLocatedLogReader

{

public:

virtual int read\_log(const int64\_t file\_id, const int64\_t offset, const int64\_t start\_log\_id, int64\_t& end\_log\_id,

char\* buf, const int64\_t len, int64\_t& read\_count);

private:

const char\* log\_dir\_;

bool dio\_; // 是否使用DirectIO

};

读取的日志不超过调用者给定的缓冲区长度，也不能越过文件末尾跳到下一个文件。读取文件成功后，接着解析一遍读到的日志，去掉缓冲区最后可能剩余的半个日志，并同时得到开始和结束的日志ID。 start\_log\_id 在调用时作为常量给出，返回之前会检查实际读到的第一条日志ID是否真的等于 start\_log\_id 。

1. 返回的日志是由 [start\_log\_id,end\_log\_id) 组成的左闭右开的区间。
2. 读取不到日志(到了文件末尾)不认为是错误，这时 read\_count=0, end\_log\_id=start\_log\_id
3. 调用者一般每次都会请求读取约2M的日志, 所以每次读取磁盘应该也不会有性能问题。

**1.2 位置缓存和实际定位： ObDefaultLogLocator 的实现**

为了根据日志ID快速定位文件ID和文件内偏移，使用两个类：

1. ObLogLocationCache 日志的位置缓存。 缓存的形式是: 日志ID->(文件ID, 文件偏移)
2. ObOnDiskLogLocator 缓存未命中时，读取磁盘文件定位日志。

这两个类都实现了 IObLogLocator 接口

**2 日志位置缓存： ObLogLocationCache**

缓存由一个固定大小的数组表示：

1. 缓存的形式是 日志ID->(文件ID, 文件偏移)，接口如下:

struct ObLogLocationCache {

int get\_location(int64\_t log\_id, ObLogLocation& location);

int enter\_location(const ObLogLocation& location);

};

1. 给定日志ID为 log\_id ，数组长度为 array\_len, 日志的位置信息在环形数组中的位置为 log\_id % array\_len ,
2. 向缓存中添加位置信息时，如果有冲突，替换规则是日志ID大的可以替换日志ID小的
3. 缓存可能会并发更新，更新时也可能有读者读取，需要保证并发读写时位置信息的完整性。

上面这些需求实际上可以由一个有明确替换优先级的KV cache完成，并且KV都是定长的，为了代码重用,缓存用一个 ObRecentCache 的模板类实现.

template<typename K, typename V>

class ObRecentCache

{

public:

//ObAtomicType保证并发读写的原子性

struct AssocItem : public ObAtomicType<AssocItem>

{

AssocItem(): k(), v() {} // k应该初始化为最小值

K k;

V v;

};

int get(const K k, V& v) const

int add(const K k, const V& v);

private:

int64\_t n\_items\_;

AssocItem\* items\_;

};

get(k, v)

1. 原子性拷贝 items\_[k % n\_items\_] 到一个临时变量 item ，若拷贝失败，返回 OB\_ENTRY\_NOT\_EXIST ；
2. 检查 item.k 是否等于 k ，若不等，返回 OB\_ENTRY\_NOT\_EXIST ; 若相等，设置好 v , 返回成功。

add(k, v)

1. 检查 items\_[k % n\_items\_].k 是否大于 k ，若前者更大，直接返回。
2. 用 k,v 初始化一个临时变量 item , 原子性更新 items\_[k % n\_items\_] 的值为 item

**3 实际定位： ObOnDiskLogLocator**

1. 根据日志ID得到文件ID，由 ObLogFileIdLocator 完成
2. 在单个日志文件内得到给定ID的日志的偏移, 由 ObLogFileOffsetLocator 完成。

**3.1 由日志ID查询文件ID**

ObLogFileIdLocator {

virtual int get\_first\_log\_id(const int64\_t file\_id, int64\_t& log\_id);

int get\_file\_id\_helper(const int64\_t max\_log\_file\_id, int64\_t log\_id, int64\_t& file\_id)

int get\_first\_log\_id(int64\_t file\_id, int64\_t& start\_log\_id); // 获取file\_id的日志文件的第一条日志ID

}

get\_file\_id\_helper(max\_log\_file\_id, log\_id, &file\_id) 实现定位文件ID的功能: 从编号最大的日志文件开始，找到第一个第一条日志ID小于等于 log\_id 的文件，返回这个文件的ID。

1. 设置 try\_file\_id 的值为 max\_log\_file\_id
2. 调用 get\_start\_log\_id(try\_file\_id, start\_log\_id) 尝试获得 try\_file\_id 对应的日志文件的第一条日志的ID， 若这条日志ID小于 log\_id , 设置 file\_id 为 try\_file\_id ，返回成功; 否则，将 try\_file\_id 减1，继续尝试，直到 try\_file\_id 对应的文件不存在，返回失败。

为完成定位, 有两个功能需要实现:

1. 如何知道磁盘上已经生成的最大日志文件ID？ 每次扫描日志目录是不可取的。 实际使用时可以在第一次启动时扫描日志目录更新一次最大日志ID，并且 log\_writer 每次新写一个日志文件更新一次最大日志ID。 这里把问题推到后面解决，留下了一个接口: IObMaxLogFileIdGetter 。
2. 如何获取每个日志文件的第一条日志的ID？ 使用 ObRecentCache 实现一个: 文件ID->第一条日志ID 的cache。 首先查询缓存，如果命中，直接返回，如果未命中，使用 ObRepeatedLogReader 打开日志文件，读取第一条日志即可, 更新缓存并返回。

**3.2 由日志ID和文件ID查询文件偏移**

ObLogFileOffsetLocator 在给定的文件ID中定位给定编号的日志，定位失败时返回 OB\_ENTRY\_NOT\_EXIST 。

class ObLogFileOffsetLocator : public IObLogFileOffsetLocator

{

public:

virtual int get\_file\_offset(const int64\_t log\_id, const int64\_t file\_id, int64\_t& offset);

private:

const char\* log\_dir\_;

};

返回的 offset 的含义是指: 从给定文件的 offset 开始读，读到的第一条日志编号为 log\_id .

get\_file\_offset(log\_id, file\_id, offset)

1. 打开编号为 file\_id 的日志文件，
2. 从当前位置读取一条日志，假设当前日志编号为 cur\_id :
   1. 如果 cur\_id 大于 log\_id, 返回错误；
   2. 如果 cur\_id 等于 log\_id-1, 设置 offset 为当前位置, 返回；
   3. 如果 cur\_id 等于 log\_id, 要读的日志是这个文件的第一条日志，设置 offset 为0, 返回；
   4. 如果 cur\_id 小于 log\_id-1, 重复第二步
   5. 如果遇到错误或读到文件末尾时退出第二步循环。

定位日志偏移时使用 ObRepeatedLogReader 打开指定的日志文件，一条一条的读取日志，由于 ObRepeatedLogReader 内部设有缓存，所以并不会每次读一条日志就读一次磁盘。

**关闭提示** [**关闭**](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html)

[**确 认**](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html) [**取 消**](http://10.232.35.40:8001/home/yuanqi.xhf/org/archive/pos-log-reader.html)