InnoDB存储引擎的四大特性

- 插入缓冲（insertbuffer)
- 二次写(doublewrite)
- 自适应哈希索引(ahi)
- 预读(readahead)

预备知识

聚集索引：聚集索引是索引结构和数据一起存放的索引。类似于字典的正文，当我们根据拼音直接就能找到那个字。

非聚集索引：非聚集索引是索引结构和数据分开存放的索引。类似于根据偏旁部首找字，首先找到该字所在的地址，再根据地址找到这个字的信息。

区别：

1.聚集索引一个表只能有一个，而非聚集索引一个表可以存在多个

2.聚集索引存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续

3.聚集索引:物理存储按照索引排序；聚集索引是一种索引组织形式，索引的键值逻辑顺序决定了表数据行的物理存储顺序。

4.非聚集索引:物理存储不按照索引排序；非聚集索引则就是普通索引了，仅仅只是对数据列创建相应的索引，不影响整个表的物理存储顺序。

5.索引是通过二叉树的数据结构来描述的，我们可以这么理解聚簇索引：索引的叶节点就是数据节点。而非聚簇索引的叶节点仍然是索引节点，只不过有一个指针指向对应的数据块。

优势与缺点：

聚集索引插入数据时速度要慢（时间花费在“物理存储的排序”上，也就是首先要找到位置然后插入），查询数据比非聚集数据的速度快。

1.使用聚集索引的查询效率要比非聚集索引的效率要高，但是如果需要频繁去改变聚集索引的值，写入性能并不高，因为需要移动对应数据的物理位置。

2.非聚集索引在查询的时候尽量避免二次查询，这样性能会大幅提升。

3.不是所有的表都适合建立索引，只有数据量大表才适合建立索引，且建立在选择性高的列上面性能会更好。

聚集索引是索引叶子节点上存的是数据，非聚集索引的叶子节点存的是数据的指针。

聚集索引是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储不连续。

聚集索引每张表只能有一个，非聚集索引可以有多个。

mysql的innodb引擎必须要有主键，因为数据存放在聚集索引上，即使不设置主键，mysql也会设置一个默认主键，需要去存放数据。其他索引也就是非聚集索引或者叫二级索引（辅助索引）存放的是主键的数据，从而根据主键的查找到数据。

myisam引擎可以不需要主键，因为引擎会单独存储数据，索引上存放的是指向数据的指针。

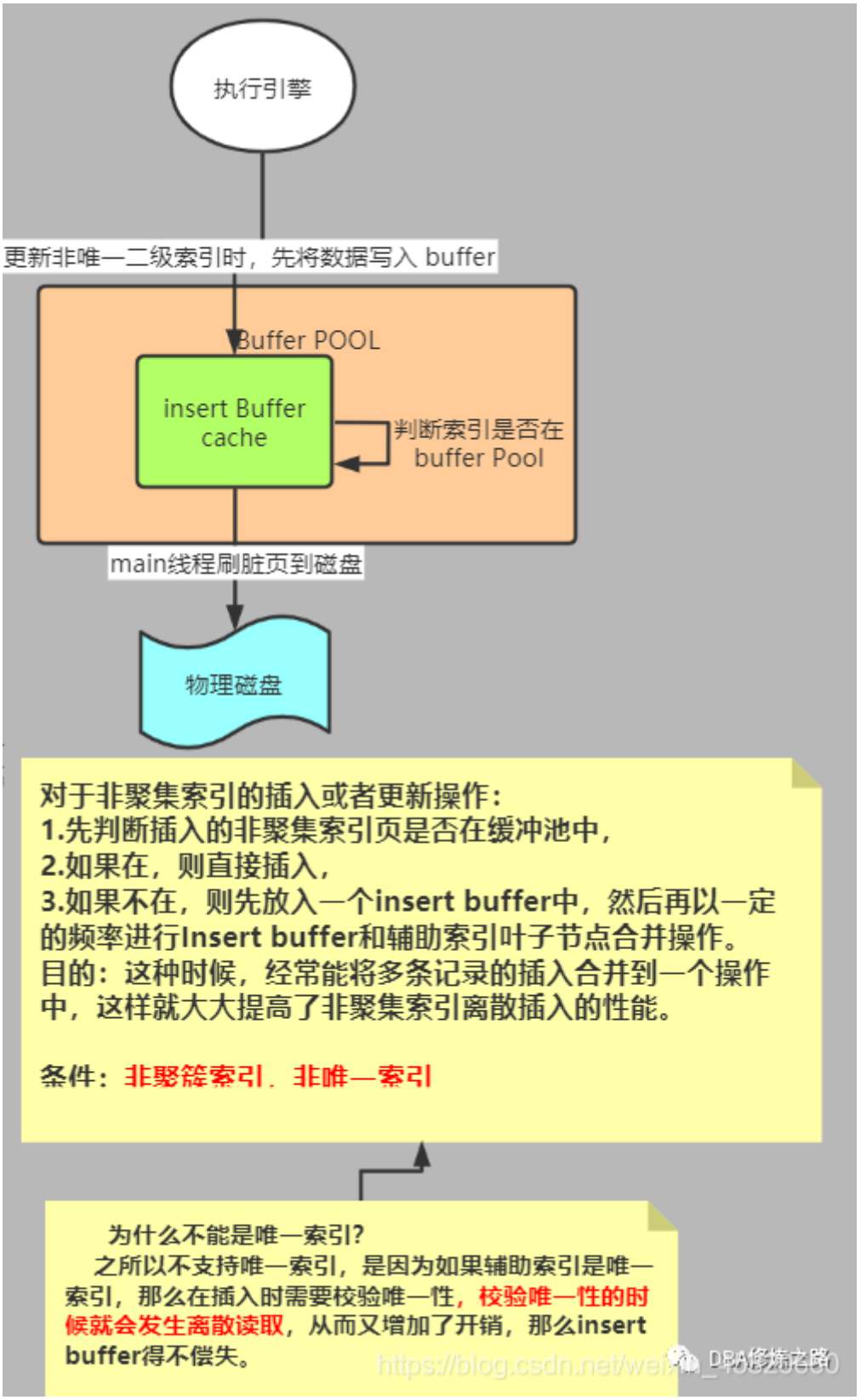
1.插入缓冲 (Insert Buffer/Change Buffer)

在insert数据时，如果该数据的非聚集索引页存在于缓冲池中，那么就直接插入到索引页中，如果不存在，则插入到insert buffer中，然后按照一定的频率进行合并操作，写入磁盘，这样做的目的就是为了能合并操作，减少磁盘的写入次数，注意只是非聚集索引是需要这样的，聚集索引（id）一般都是自增的，写入的位置都是顺序的，所以效率很高，不需要这个，但是非聚集索引就等于是随机写，效率较低。

插入缓存之前版本叫insert buffer，现版本 change buffer，主要提升插入性能，change buffer是insert buffer的加强，insert buffer只针对insert有效，change buffering对insert、delete、update(delete+insert)、purge都有效。有什么用呢？

对于非聚集索引来说，比如存在用户购买金额这样一个字段，索引是普通索引，每个用户的购买的金额不相同的概率比较大，这样导致可能出现购买记录在数据里的排序可能是1000，3，499，35…，这种不连续的数据，一会插入这个数据页，一会插入那个数据页，这样造成的IO是很耗时的，所以出现了Insert Buffer。

Insert Buffer是怎么做的呢？mysql对于非聚集索引(非聚集索引是一种索引，该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同。)的插入，先去判断要插入的索引页是否已经在内存中了，如果不在，暂时不着急先把索引页加载到内存中，而是把它放到了一个Insert Buffer对象中，临时先放在这，然后等待情况，等待很多和现在情况一样的非聚集索引，再和要插入的非聚集索引页合并，比如说现在Insert Buffer中有1，99，2，100，合并之前可能要4次插入，合并之后1，2可能是一个页的，99，100可能是一个页的，这样就减少到了2次插入。这样就提升了效率和插入性能，减少了随机IO带来性能损耗。



上面提过在一定频率下进行合并，那所谓的频率是什么条件？

1）辅助索引页被读取到缓冲池中。正常的select先检查Insert Buffer是否有该非聚集索引页存在，若有则合并插入。

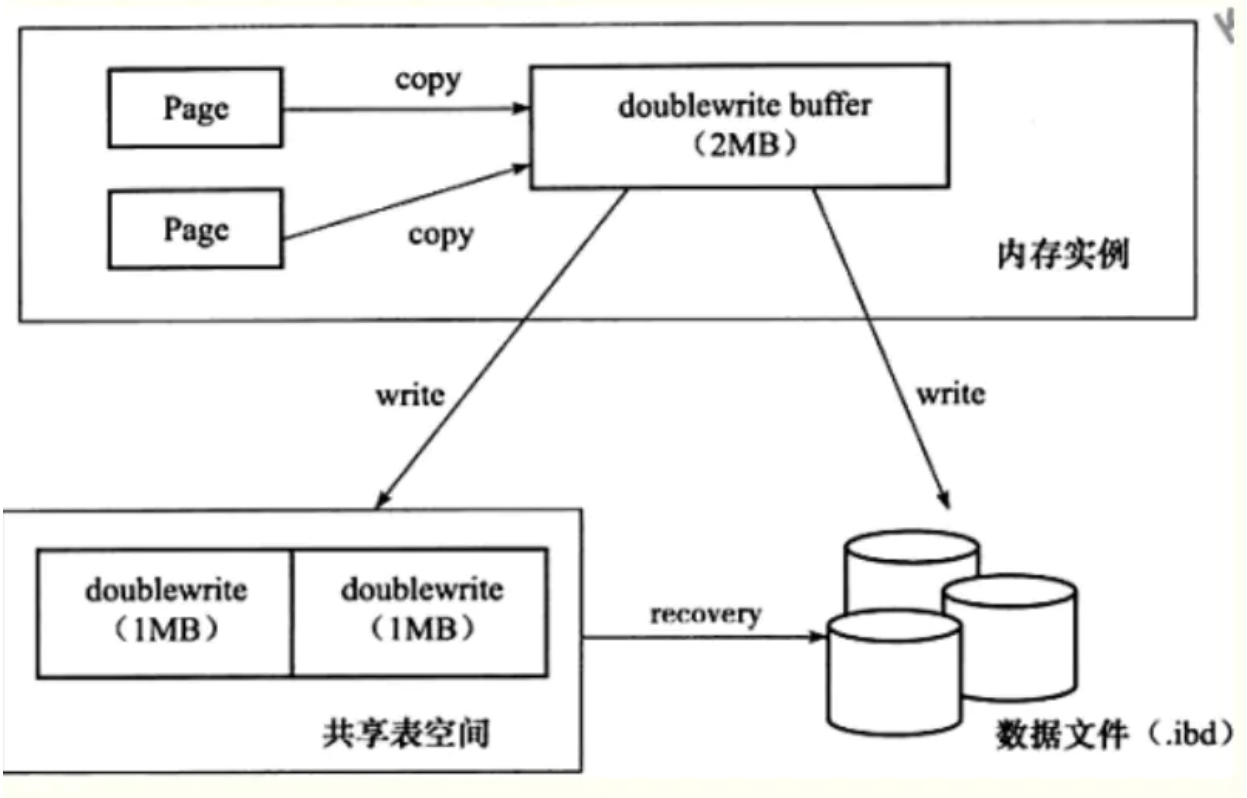
2）辅助索引页没有可用空间。空间小于1/32页的大小，则会强制合并操作。

3）Master Thread 每秒和每10秒的合并操作。

2.二次写（Double Write)

innodb页的大小是16k，相对来说还是比较大的，所以当将脏页写回到磁盘中时，可能发生断电宕机等问题，导致写入了一半，这个时候就没法恢复了，所以使用了两次写这样的机制

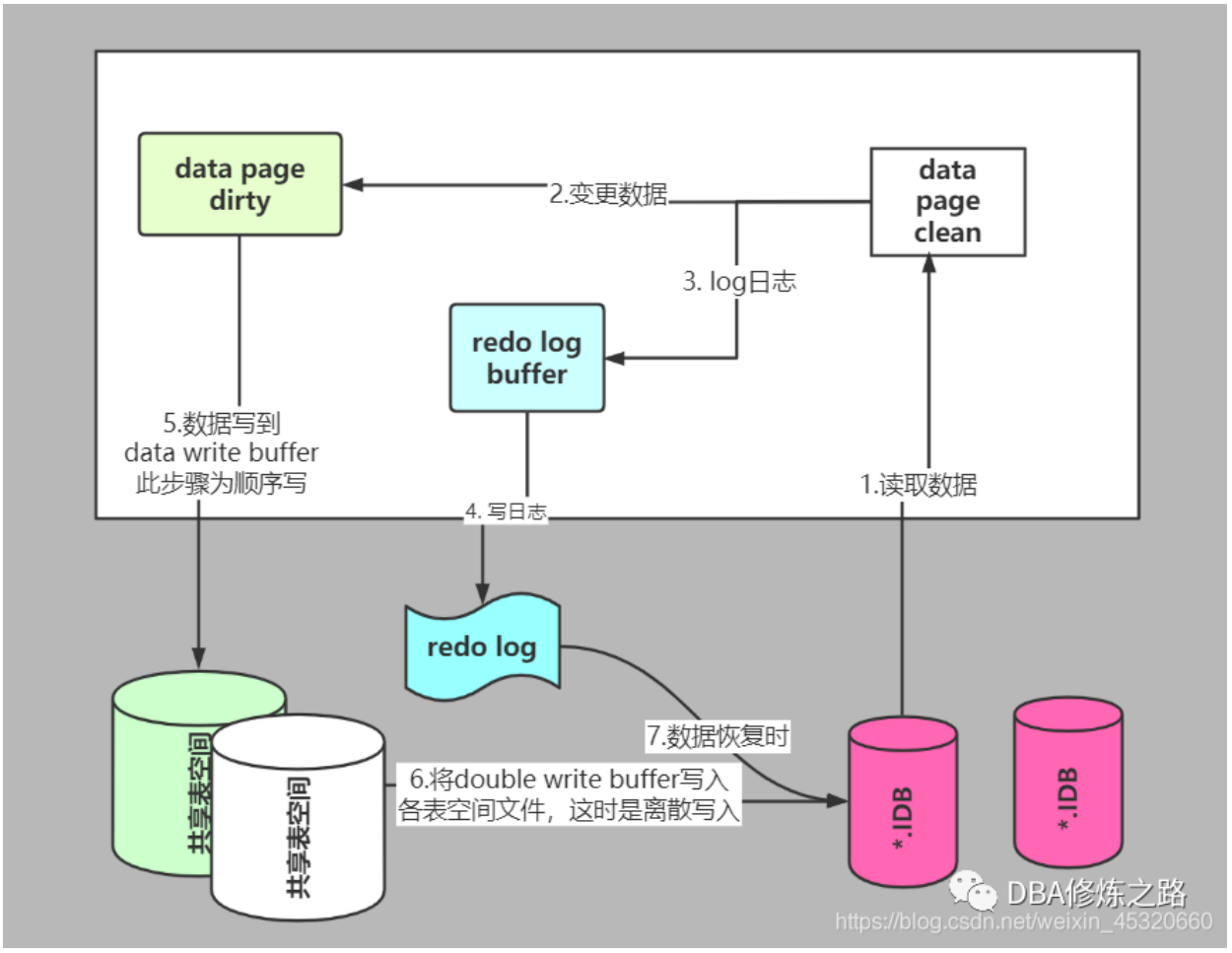
当页需要写回数据库时，首先把页备份到内存中的doublewrite buffer，然后每次1M，写入到共享表空间中，共享表空间也是在磁盘上，因为是顺序写，所以很快，然后再将这些页写入到真的数据文件中，就算这个时候服务器出了问题，也是可以用共享表空间中的数据进行还原的



在InnoDB将BP中的Dirty Page刷（flush）到磁盘上时，首先会将（memcpy函数）Page刷到InnoDB tablespace的一个区域中，我们称该区域为Double write Buffer（大小为2MB，每次写入1MB，128个页，每个页16k,其中120个页为后台线程的批量刷Dirty Page，还有8个也是为了前台起的sigle Page Flash线程，用户可以主动请求，并且能迅速的提供空余的空间）。在向Double write Buffer写入成功后，第二步、再将数据分别刷到一个共享空间和真正应该存在的位置。

MySQL可以根据redolog进行恢复,而mysq在恢复的过程中是检查page"的checksum, checksum就是pgae的最后事务号,发生partial page write问题时. DageR经损坏,找不到该page中的事务号就无法恢复。

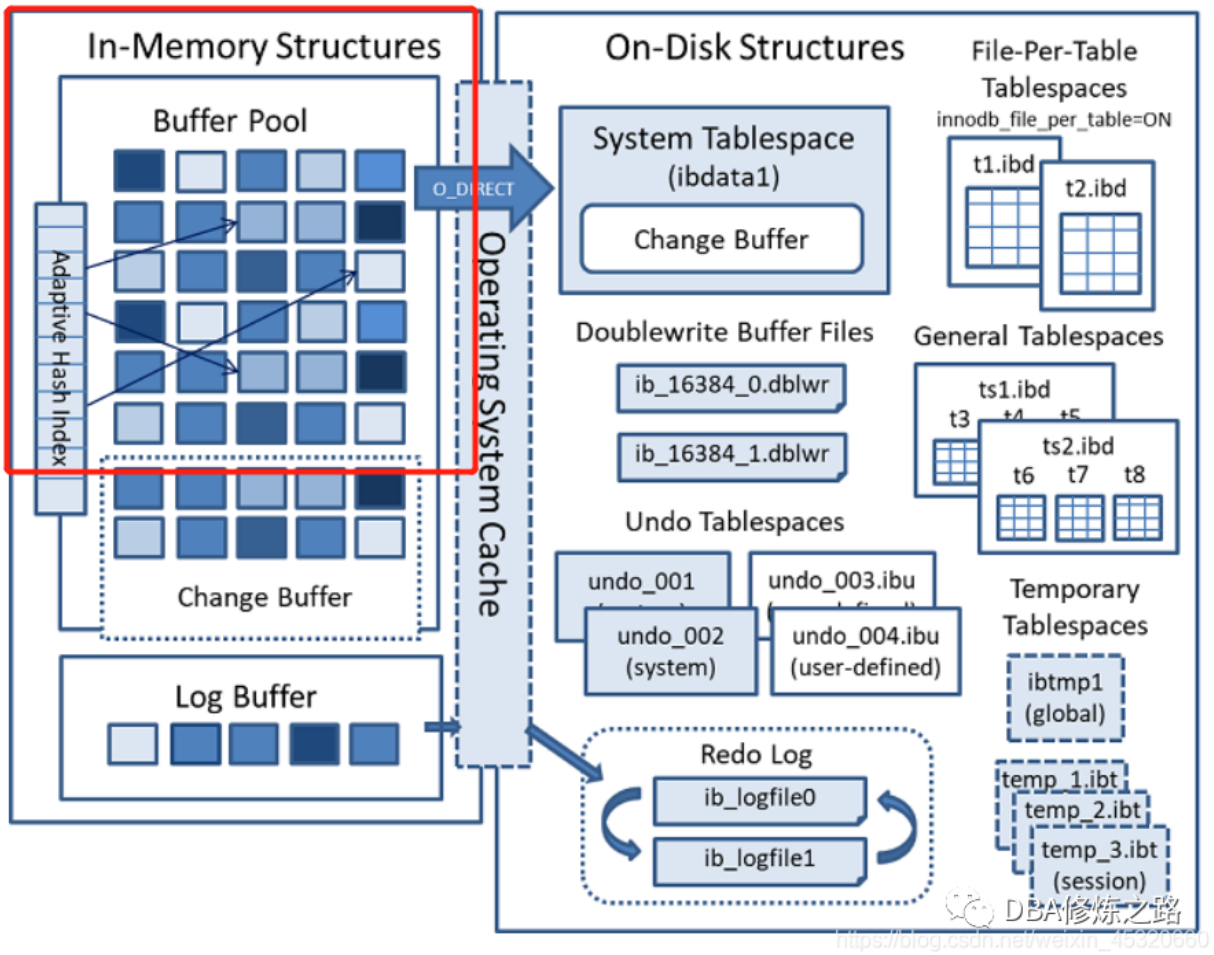
具体的流程如下图所示：



3.自适应哈希索引（Adaptive Hash Index，AHI)

当某个非聚集索引被等值查询的次数很多时，就会为这个非聚集索引再构造一个hash索引，hash索引对呀等值查询是很快的，这个hash索引会放在缓存中

哈希算法是一种非常快的查找方法，在一般情况（没有发生hash冲突）下这种查找的时间复杂度为O(1)。InnoDB存储引擎会监控对表上辅助索引页的查询。如果观察到建立hash索引可以提升性能，就会在缓冲池建立hash索引，称之为自适应哈希索引（Adaptive Hash Index，AHI）。



自适应哈希索引由innodb_adaptive_hash_index 变量启用,AHI是通过缓冲池的B+ Tree构造而来，使用索引键的前缀来构建哈希索引，前缀可以是任意长度。InnoDB存储引擎会自动根据访问的频率和模式来自动地为某些热点页建立hash索引。加快索引读取的效果，相当于索引的索引，帮助InnoDB快速读取索引页。

4.预读 （Read Ahead)

innodb中将64个页划分为一个extent，当一个extent中的页，被顺序读超过了多少个，比如50个，这个值是可以通过nnodb_read_ahead_threshold设置的，那么就会认为顺序读到下一个extent的可能性很大，会提前将下一个extent中的所有页都加载到buffer pool中，这叫线性预读

如果某一个extent中，有多个页被读到，就会认为读到这个extent中其他页的可能性也很大，就会把该extent中的其他页也都提前读到buffer pool中

预读（read-ahead)操作是一种IO操作，用于异步将磁盘的页读取到buffer pool中，预料这些页会马上被读取到。预读请求的所有页集中在一个范围内。InnoDB使用两种预读算法：

Linear read-ahead：线性预读技术预测在buffer pool中被访问到的数据它临近的页也会很快被访问到。能够通过调整被连续访问的页的数量来控制InnoDB的预读操作，使用参数innodb_read_ahead_threshold配置，添加这个参数前，InnoDB会在读取到当前区段最后一页时才会发起异步预读请求
innodb_read_ahead_threshold 这个参数控制InnoDB在检测顺序页面访问模式时的灵敏度。如果在一个区块顺序读取的页数大于或者等于 innodb_read_ahead_threshold 这个参数，InnoDB启动预读操作来读取下一个区块。innodb_read_ahead_threshold参数值的范围是 0-64，默认值为56. 这个值越高则访问默认越严格。比如，如果设置为48，在当前区块中当有48个页被顺序访问时，InnoDB就会启动异步的预读操作，如果设置为8，则仅仅有8个页被顺序访问就会启动异步预读操作。你可以在MySQL配置文件中设置这个值，或者通过SET GLOBAL 语句动态修改（需要有set global 权限）。

Random read-ahead: 随机预读通过buffer pool中存中的来预测哪些页可能很快会被访问，而不考虑这些页的读取顺序。如果发现buffer pool中存中一个区段的13个连续的页，InnoDB会异步发起预读请求这个区段剩余的页。通过设置 innodb_random_read_ahead 为ON开启随机预读特性。
通过 SHOW INNODB ENGINE STATUS 命令输出的统计信息可以帮助你评估预读算法的效果，统计信息包含了下面几个值：

innodb_buffer_pool_read_ahead 通过预读异步读取到buffer pool的页数
innodb_buffer_pool_read_ahead_evicted 预读的页没被使用就被驱逐出buffer pool的页数，这个值与上面预读的页数的比值可以反应出预读算法的优劣。
innodb_buffer_pool_read_ahead_rnd 由InnoDB触发的随机预读次数。