

MySQL——Buffer Pool

○、大纲

一、缓存的重要性

二、Buffer Pool概述

三、几种主要的链表

3.1> free链表

3.2> flush链表

3.3> LRU链表

四、其他补充知识点

4.1> 多个Buffer Pool实例

4.2> chunk

4.3> 配置Buffer Pool时的注意事项

○、大纲

- BufferPool 的作用是什么？
- BufferPool 的初始默认大小是多少？
- 缓冲页 的大小是多少？
- 控制块 和 缓冲页 是什么关系？
- 如何判断一个页是否在 BufferPool 中被缓存了？
- 什么是 free链表 ？
- 什么是 flush链表 ？
- 刷脏页 的方式有哪些？
- 什么是 LRU链表 ？
- 什么是 预读 ？
- 什么是 全表扫描 ？
- LRU 如何应对预读和全表扫描对缓存数据的“破坏”？

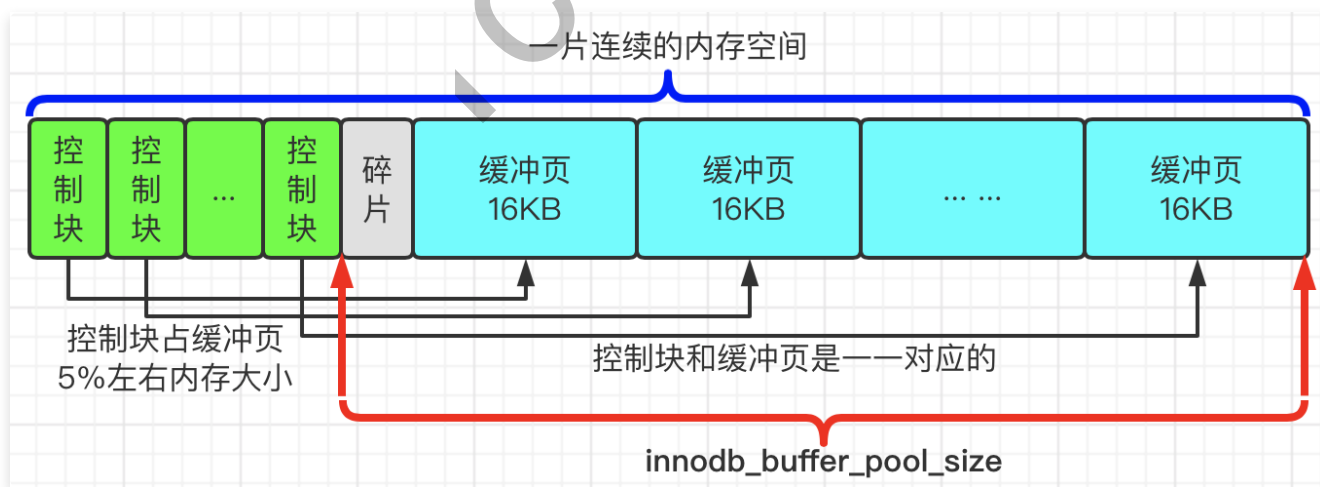
一、缓存的重要性

- 无论是用于存储用户数据的索引，还是各种系统数据，都是以页的形式存放在表空间中的。

- 所谓**表空间**，只不过是InnoDB对一个或几个**实际文件的抽象**。也就是说，我们的数据说到底还是存储在**磁盘**上的。
- 但是磁盘读取速度很慢，所以如果需要访问某个页的数据时，InnoDB会把**完整的页**中的数据**全部加载到内存**中。即使只需要访问一个页里面的一条记录，也需要先把整个页的数据加载到内存中。然后就可以在内存中对记录进行读写访问了。
- 在读写访问之后，并不着急把该页对应的内存空间释放掉，而是将其缓存起来，如果将来再次访问该页面，就可以减少I/O的开销了。

二、Buffer Pool概述

- 为了缓存磁盘中的页，MySQL服务器**启动时**就向操作系统申请了一片**连续**的内存空间，他们给这片内存起名为—— **Buffer Pool**（缓冲池）。
- 默认 **Buffer Pool** 只有**128M**，可以在启动服务器的时候配置**innodb_buffer_pool_size**（单位为字节）启动项来设置自定义缓冲池大小。
- Buffer Pool对应的一片**连续的内存**被划分为若干个页面，默认也是**16KB**。该页面称为**缓冲页**。
- 为了更好的**管理**Buffer Pool中的这些缓冲页，InnoDB为每个缓冲页都创建了**控制块**。它与缓冲页是一一对应的。
- **控制块**存放到Buffer Pool的**前面**，**缓冲页**存放到Buffer Pool的**后面**，如下所示：



- **如何判断一个页是否在Buffer Pool中被缓存了？**

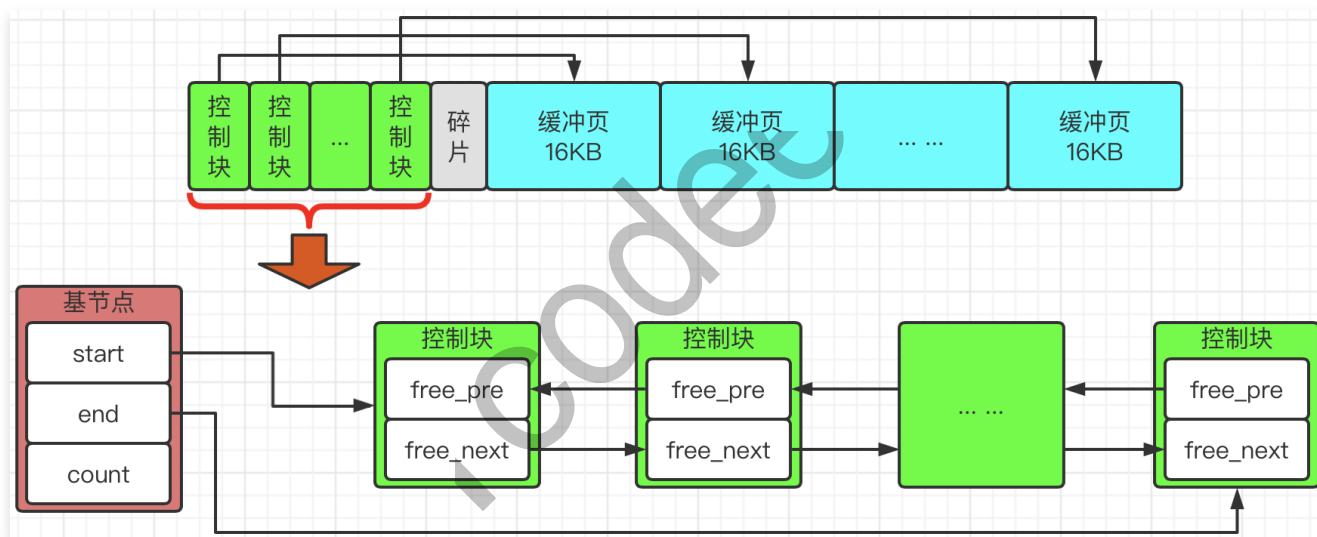
其实是存在缓冲页的**哈希表**的。key= **表空间号+页号** value= **缓冲页控制块**。所以，当需要访问某个页的数据时，先从哈希表中根据表 **空间号+页号** 看看是否存在对应的缓冲页。如果有，则直接使

用；如果没有，就从**free链表**中选出一个空闲的缓冲页，然后把磁盘中对应的也加载到该缓冲页的位置。

三、几种主要的链表

3.1> free链表

- Buffer Pool的初始化过程中，是先向操作系统**申请连续的内存空间**，然后把它划分成若干个**【控制块&缓冲页】**对儿。
- 当插入数据的时候，为了能够知道哪些缓冲页是**空闲可分配**的，由此产生了free链表。
- free链表是**把所有空闲的缓冲页**对应的**控制块**作为一个节点放到一个链表中，这个链表便称之为free链表。
- free链表如下所示：



其中**基节点**是一块单独申请的内存空间（约占**40字节**）。并不在Buffer Pool的那一大片连续内存空间里。

- 磁盘加载页的流程

首先：从free链表中取出一个空闲的控制块（对应缓冲页）。

其次：把该缓冲页对应的控制块的信息填上（例如：页所在的表空间、页号之类的信息）。

最后：把该缓冲页对应的free链表节点（即：控制块）从链表中移除。表示该缓冲页已经被使用了。

3.2> flush链表

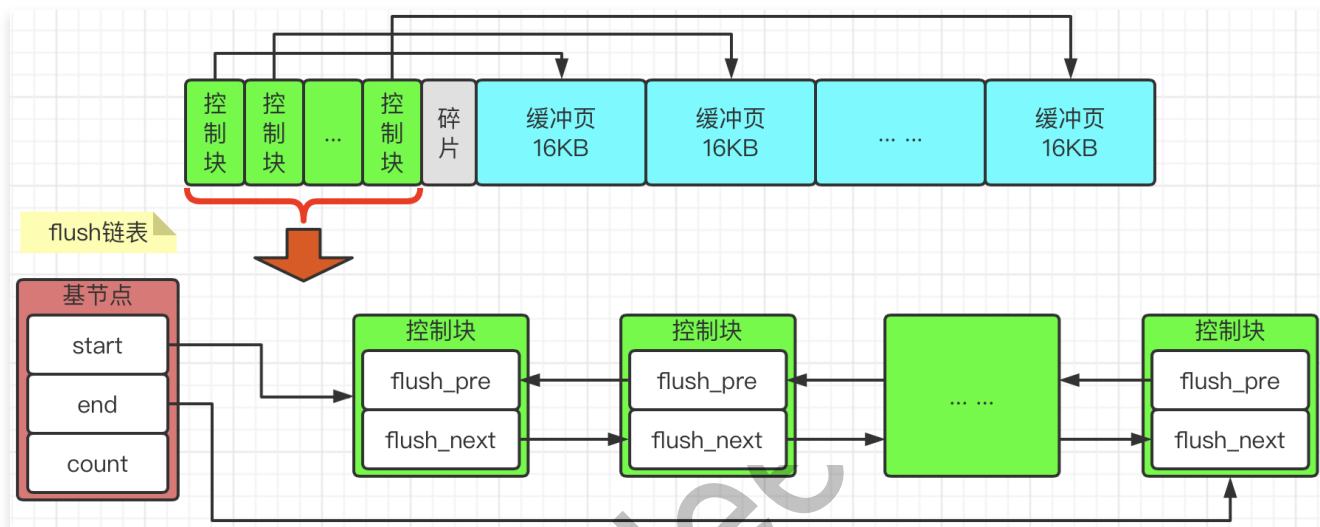
- 如果我们**修改了**Buffer Pool中某个缓冲页的数据，那么它就**与磁盘上的页不一致了**，这样

的缓冲页也被称之为**脏页**（dirty page）。为了性能问题，我们每次修改缓冲页后，并不着急立刻把修改刷新到磁盘上，而是在**未来的某个时间点进行刷新操作**。

- 那么，如果有了修改发生，不是立刻刷新，那之后再刷新的时，我们怎么知道Buffer Pool中哪些页是脏页，哪些页从来没有被修改过呢？

答：创建一个存储脏页的链表，凡是被修改过的缓冲页对应的**控制块**都会作为节点加入到这个链表中。该链表也被称为flush链表。

- flush链表的结构与free链表差不多。flush链表如下所示：



- 那么，会存在一个控制块既是free链表的节点，也是flush链表的节点吗？

答：不会的。因为如果一个缓冲页是空闲的，那它肯定不可能是脏页。反之亦然。

- 后台有**专门的线程**负责每隔一段时间就把脏页刷新到磁盘。这样就不影响用户线程处理正常的请求了。
- 刷新方式有如下**两种**：

1> 从**flush链表**中刷新一部分页面到磁盘

- 后台线程会根据当时**系统的繁忙程度**确定刷新速率，**定时**从flush链表中刷新一部分页面到磁盘。——即： `BUF_FLUSH_LIST`
- 有时后台线程刷新脏页的进度比较慢，导致用户准备加载一个磁盘页到Buffer Pool中时**没有可用的缓冲页**。此时，就会尝试查看**LRU链表尾部**，看是否存在可以直接释放掉的**未修改**缓冲页。如果没有，则不得不**将LRU链表尾部的一个脏页同步刷新到磁盘**（与磁盘交互是很慢的，这会降低处理用户请求的速度）。——即： `BUF_FLUSH_SINGLE_PAGE`

2> 从**LRU链表的冷数据**中刷新一部分页面到磁盘——即： `BUF_FLUSH_LRU`

- 后台线程会**定时**从LRU链表的**尾部**开始扫描一些页面，扫描的页面数量可以通过系统变量 `inn`

`odb_lru_scan_depth` 来指定，如果在LRU链表中发现脏页，则把它们刷新到磁盘。

- 控制块里会存储该缓冲页是否被修改的信息，所以在扫描LRU链表时，可以很轻松地获取到某个缓冲页是否是脏页的信息。

3.3> LRU链表

- LRU = Least Recently Used
- 由于缓冲区空间有限，如果满了，则需要把旧的移除掉，新的加进来。那么移除规则是什么呢？
- 我们的目的，肯定是为了把使用频繁的数据保留在缓存中，把使用频率低的数据移除。
- 用来记录缓冲页的被使用热度。
- Buffer Pool的缓冲命中率（我们当然是期望命中率越高越好）

假设我们一共访问了n次页，那么被访问的页已经在Buffer Pool中的次数除以n，那么就是Buffer Pool的缓冲命中率。

- 提高命中率的方法——简单的LRU链表

【处理逻辑】

1> 创建LRU链表。

2> 当要访问某个页时，如果不在Buffer Pool，则把该页从磁盘加载到缓冲池的缓冲页时，就把该缓冲页对应的控制块作为节点塞到LRU链表的头部。

3> 如果在Buffer Pool中，则直接把该页对应的控制块移动到LRU链表的头部。

【方案优点】

所有最近使用的数据都在链表表头，最近未使用的数据都在链表表尾。

【方案缺点】

1> 由于预读（下面会介绍）的行为，很多预读的页都会被放到LRU链表的表头。如果这些预读的页都没有用到的话，这样，会导致很多尾部的缓冲页很快就会被淘汰。

2> 如果发生全表扫描（比如：没有建立合适的索引 or 查询时没有where字句），则相当于把原有缓冲页全部都冲刷没了。

- 什么是预读？
 - 就是InnoDB认为执行当前的请求时，可能会在后面读取某些页面，于是就预先把这些页面加载到Buffer Pool中。
 - 根据触发方式不同，预读可以分为两种：

- 线性预读

如果**顺序访问**某个区（extent，一个区默认**64**个页）的页面超过了 **innodb_read_ahead_threshold**（默认 **56**）的值，就会触发一次异步读取 **下一个区 中全部的页**到Buffer Pool中的请求。

- 随机预读

如果开启了随机预读功能（默认**innodb_random_read_ahead=OFF**），如果某个区（extent）有 **13 个连续的页面** 已经被加载到了Buffer Pool中，**无论这些页面是不是顺序读取的**，都会触发一次异步读取 **本区 全部的页**到Buffer Pool中的请求。

- 综上所述，其实造成Buffer Pool命中率低有两种情况：

1> 加载到Buffer Pool中的页**不一定被用到**。

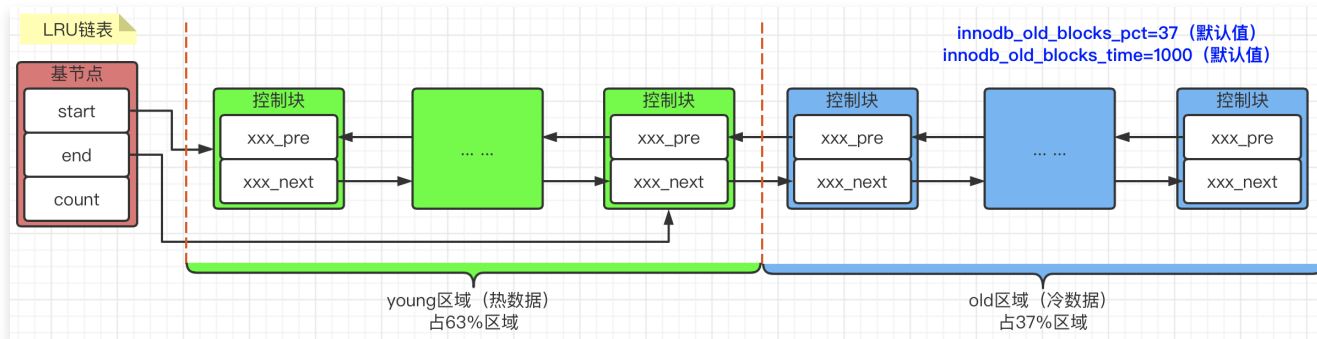
2> 如果有非常多的使用频率偏低的页被同时加载到Buffer Pool中，则可能会把那些使用**频率非常高的页**从Buffer Pool中**淘汰掉**。

- 为了解决命中率低的问题，InnoDB把LRU链按比例（**innodb_old_blocks_pct**）分成了两段——**young区域**和**old区域**。

```
11 • show variables like 'innodb_old_blocks_pct';
```

Variable_name	Value
innodb_old_blocks_pct	37

- LRU链表如下所示：



- 针对简单LRU链表方案缺点的优化

1> 针对预读的优化

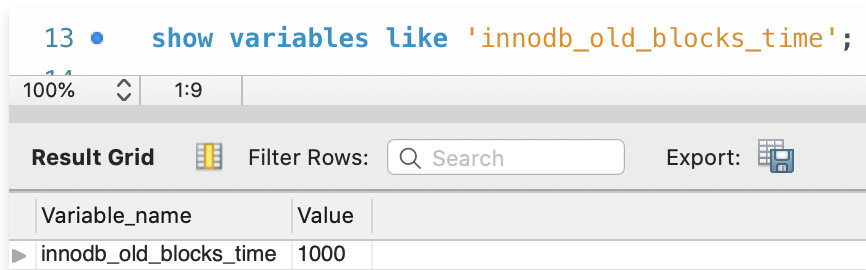
InnoDB规定，当磁盘上的某个页在**初次加载**（只是加载，没有涉及读取）到Buffer Pool中的某个缓冲页时，该缓冲页对应的控制块会被放到**old区域的头部**。这样预读页就只会在old区域，不会影

响young区域中使用比较频繁的缓冲页。

2> 针对全表扫描的优化

虽然首次加载放到的是old区域的头部，但是由于是全表扫描，会对加载的数据进行访问，那么**第一次访问**的时候，就会将该页放到young区域的头部。这样仍然会把那些使用频率比较高的页面给“排挤”下去。

那怎么办呢？由于全表扫描有一个特点，就是**它对某个页的频繁访问**且总耗时很短。所以，针对这种情况，InnoDB规定，在对某个处于**old区域**的缓冲页进行**第一次访问**时，就在它对应的**控制块**中记录下这个访问时间，如果后续的访问时间与第一次访问的时间在某个时间间隔内（即：**innodb_old_blocks_time**，默认为1000，单位为ms），那么该页面就不会从old区域移动到young区域的头部，否则将它移动到young区域的头部。



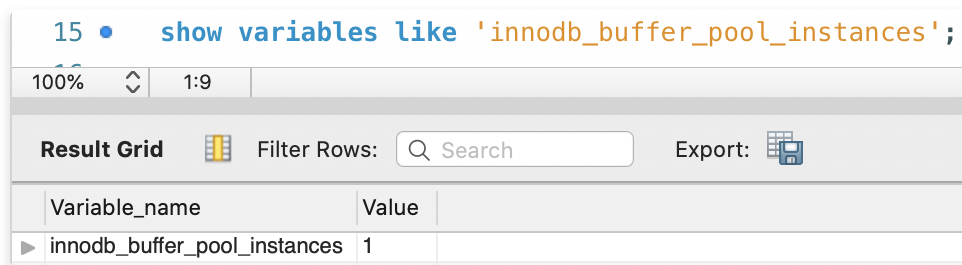
```
13 • show variables like 'innodb_old_blocks_time';
```

Variable_name	Value
innodb_old_blocks_time	1000

四、其他补充知识点

4.1> 多个Buffer Pool实例

- 在Buffer Pool特别大并且**多线程并发访问量**特别高的情况下，单一的Buffer Pool可能会影响请求的处理速度。所以，在Buffer Pool特别大时，可以把它们拆分成若干个小的Buffer Pool，每个Buffer Pool都称为一个**实例**。它们都是独立的——独立地申请内存空间，独立地管理各种链表。
- 可以通过设置**innodb_buffer_pool_instances**的值来修改Buffer Pool实例的个数



```
15 • show variables like 'innodb_buffer_pool_instances';
```

Variable_name	Value
innodb_buffer_pool_instances	1

- 每个Buffer Pool实例实际占用多少内存空间呢？

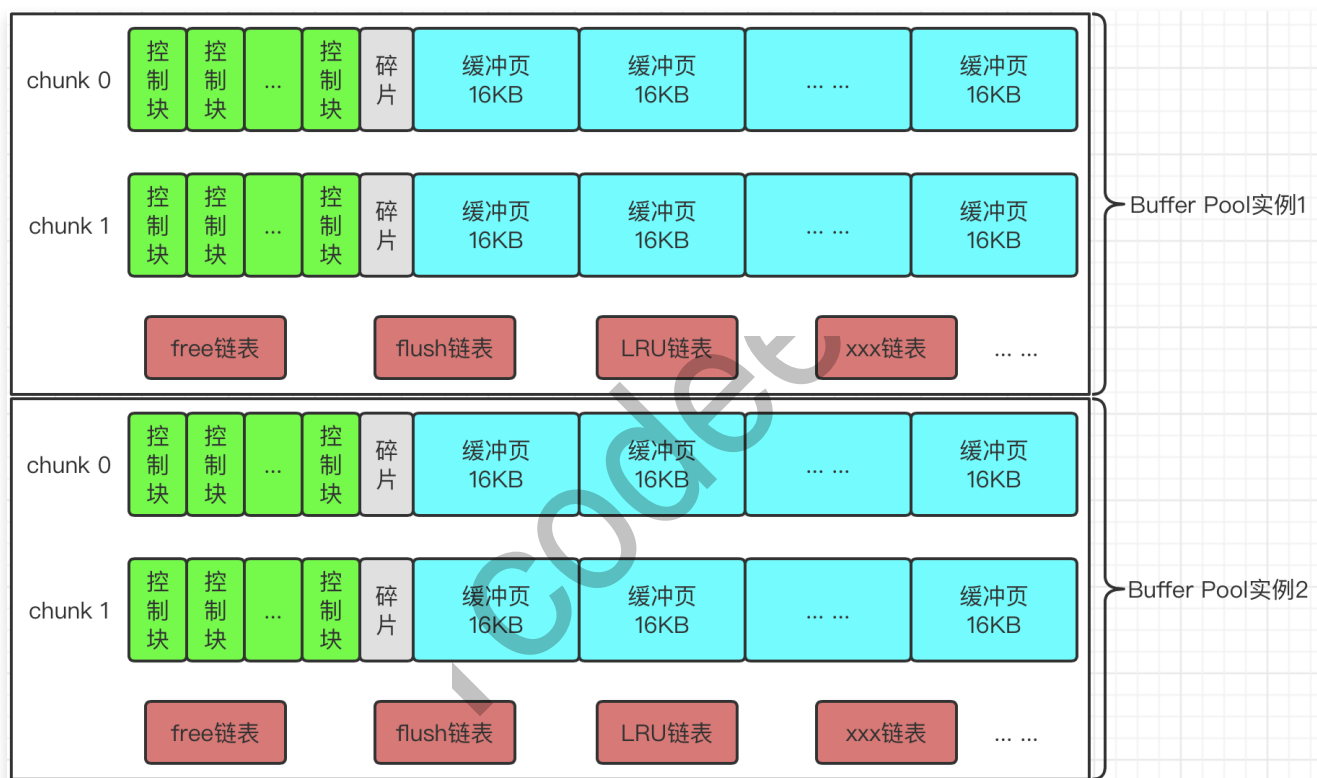
innodb_buffer_pool_size ÷ innodb_buffer_pool_instances

- 由于管理Buffer Pool也是需要性能开销的，所以也并不是实例越多越好；
- InnoDB规定，当**innodb_buffer_pool_size**小于1GB时，设置多个实例是无效的，在这种

情况下，即使你设置的`innodb_buffer_pool_instances`不为1，那么InnoDB默认也会把它改为1。

4.2> chunk

- 由于每次**调整Buffer Pool的大小**时，都需要重新向操作系统申请一块连续的内存空间，然后将旧的Buffer Pool中的内容复制到这一块新空间，但是这种操作是极其耗时的。所以，InnoDB不再一次性为某个Buffer Pool实例向操作系统申请一大片连续的内存空间，而是以一个**chunk为单元**向操作系统申请空间。也就是说，一个Buffer Pool实例其实是由若干个chunk组成的。一个**chunk就代表一片连续的内存空间**，里面包含了若干缓冲页与其对应的控制块。
- 如图所示，Buffer Pool包含2个实例，每个实例里包含2个chunk：



- 可以通过`innodb_buffer_pool_chunk_size`指定chunk的大小

```
17 • show variables like 'innodb_buffer_pool_chunk_size';
```

Variable_name	Value
innodb_buffer_pool_chunk_size	134217728

【注】

$134217728 / 1024 / 1024 = 128\text{MB}$

`innodb_buffer_pool_chunk_size`的值并不包含缓冲页对应的控制块的内存空间大小。

4.3> 配置Buffer Pool时的注意事项

- innodb_buffer_pool_size必须是innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances的倍数。否则服务器会自动把innodb_buffer_pool_size的值调整为【innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances】结果的整数倍。

例如：innodb_buffer_pool_chunk_size=128MB

innodb_buffer_pool_instances=16

则：innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances=2GB

如果我们设置innodb_buffer_pool_size=9GB，则会被自动调整为10GB

- 在服务启动时，如果innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances > innodb_buffer_pool_size的值，那么innodb_buffer_pool_chunk_size的值会被服务器自动设置为innodb_buffer_pool_size ÷ innodb_buffer_pool_instances的值。

例如：innodb_buffer_pool_chunk_size=256MB

innodb_buffer_pool_instances=16

则：innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances=4GB

如果我们设置innodb_buffer_pool_size=2GB，因为2GB < 4GB，则innodb_buffer_pool_chunk_size被修改为128MB。

吾尝终日而思矣，不如须臾之所学也；
吾尝跂而望矣，不如登高之博见也。
登高而招，臂非加长也，而见者远；
顺风而呼，声非加疾也，而闻者彰。
假舆马者，非利足也，而致千里；
假舟楫者，非能水也，而绝江河。
君子生非异也，善假于物也。

----- 摘自《劝学》

愿本文可以成为大家的“山”、“风”、“马”、“舟”，助大家在技术之路上乘风破浪，大展宏图~~
同时，也欢迎大家关注我的公众号“[爪哇缪斯](#)”~\(^o^)/~ 「干货分享，每天更新」



微信搜一搜



爪哇繆斯

codee