PTmalloc	TCmalloc	Jemalloc	
内存组织	（1）内存分配单位为chunk； （2）小于64B的chunk放在fast bin中； （3）64 - 512B放在small bin中； （4）512B - 128 KB放large bin中； （5）大于128KB不进行缓存； （6）合并后的chunk放在unsorted bin中；	（1）内存有三层缓存：PageHeap、CentralCache和ThreadCache； （2）0 - 256KB小对象放在中央缓存和线程缓存中，分为84个不同大小类别，中央缓存多个线程共享，线程级缓存每个线程私有； （3）256KB - 1MB的中对象和1MB以上大对象放在PageHeap，每个page大小为8KB；	（1）小类区间为[8B, 14kb]，共232个小类，每个类的大小并不都是2的次幂； （2）大类区间为[16kB, 7EiB]，page大小为4KB，从4 * page开始； （3）内存分配单位为extent，每个extent大小为N * 4KB，一个 `extent` 可以用来分配一次 `large_class` 的内存申请，但可以用来分配多次 `small_class` 的内存申请。
分配流程	fast bin —> small bins —> unsorted bin —> large bin —> top chunk —> 增加top chunk(sbrk/mmap) 或者 mmaped chunk；	（1）小对象：ThreadCache —> CentralCache —> PageHeap —> 内核； （2）中对象和大对象：PageHeap —> 内核；	（1）小内存：cache_bin -> slab -> slabs_nonfull -> extents_dirty -> extents_muzzy -> extents_retained -> 内核 （2）大内存：extents_dirty -> extents_muzzy -> extents_retained -> 内核
多线程支持	没有线程级缓存，每个线程进行内存分配和释放时，需要对分配区进行加锁	每个线程拥有线程级缓存，当进行小对象分配和释放时，不用加锁处理	每个线程拥有线程级缓存tcache，进行小内存分配和释放时，不用加锁
优点	它是一个标准实现，所以兼容性较好	（1）在多线程场景下，小对象内存申请和释放是无锁的，效率很高，中对象和大对象申请使用自旋锁； （2）ThreadCache会阶段性的回收内存到CentralCache里，解决了ptmalloc2中分	（1）采用多个arena来避免线程同步，多线程的分配是无锁的； （2）细粒度的锁，比如每一个bin以及每一个extents都有自己的锁，并发度更高； （3）使用了低地址优先的策略，来降低内存碎片化；

		配区之间不能迁移的问题； （3）占用更少的额外空间。例如，分配N个8字节对象可能要使用大约8N * 1.01字节的空间，即，多用百分之一的空间；	
缺点	（1）管理长周期内存时，会导致内存爆增，因为与top chunk 相邻的 chunk 不能释放，top chunk 以下的 chunk 都无法释放； （2）内存不能从一个分配区移动到另一个分配区， 就是说如果多线程使用内存不均衡，容易导致内存的浪费； （3）如果线程数量过多时，内存分配和释放时加锁的代价上升，导致效率低下； （4）每个chunk需要8B的额外空间，空间浪费大	（1）对齐操作比PTmalloc多浪费一些内存，有点空间换时间； （2）如果多个线程频繁分配大对象，对自旋锁的竞争会很激烈；	（1）arena之间的内存不可见，导致两个arena的内存出现大量交叉从而无法合并； （2）大概需要2%的额外开销，tcmalloc是1%；
适用场景	不适合多线程场景和需要申请长周期内存，只适合线程数较少和申请短周期内存的场景	适合多线程的场景	适合多线程的场景，多线程并发度更好