[转]CPU缓存一致性协议入门

作者：skyesx
原文链接：https://ld246.com/article/1467299315553
来源网站：链滴

<p style="box-sizing: border-box; margin-top: 0px !important; margin-bottom: 16px; color: #33333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="1">原文连接:http://fgiesen.wordpress.com/2014/7/07/cache-coherency/</p>
<h2 style="box-sizing: border-box; padding-bottom: 0.3em; border-bottom-width: 1px; border-bottom-style: solid; border-bottom-color: #eeeeee; margin-top: 1em; margin-bottom: 16px; font-weight: bold; line-height: 1.225; font-size: 1.75em; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-style: normal; font-variant: normal; letter-spacing: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="3">缓存（Cache）</h2>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="5">本文是关于CPU缓存的快速入门。我假设你已有了基本概念，但你可能不熟悉其中的一些细节。（如果你已经熟悉了，你可以忽略这部分。）</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="7">在现代的CPU（大多数）上，所有的内存访问需要通过层层的缓存来进行。也有些例外，比如，对映射成内存地址的I/O口、写合并（Write-combied）内存，这些访问至少会绕开这个流程的一部分。但这两者都是罕见的场景（意味着绝大多数的用态代码都不会遇到这两种情况），所以在本文中，我将忽略这两者。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="9">CPU的读/写（以及取指令）单元正常情况下至都不能直接访问内存——这是物理结构决定的；CPU都没有管脚直接连到内存。相，CPU和一级缓存（L1 Cache）通讯，而一级缓存才能和内存通讯。大约二十年前，一级缓存可以直和内存传输数据。如今，更多级别的缓存加入到设计中，一级缓存已经不能直接和内存通讯了，它和级缓存通讯——而二级缓存才能和内存通讯。或者还可能有三级缓存。你明白这个意就行。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="11">缓存是分“段”（line）的，一段对应一块存储空间，大小是32（较早的ARM、90年代/2000年代早期的x86和PowerPC）、64（新的ARM和x86）或128（较新的Power ISA机器）字节。每个缓存段知道自己对应什么范围的物理存地址，并且在本文中，我不打算区分物理上的缓存段和它所代表的内存，这听起来有点草率，但是了方便起见，还是请熟悉这种提法。具体地说，当我提到“缓存段”的时候，我就是指段和缓存大小对齐的内存，不关心里面的内容是否真正被缓存进去（就是说保存在任何级别的缓存中

了。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon
-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', '
egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo
t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t
xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we
kit-text-stroke-width: 0px;" data-source-line="13">当CPU看到一条读内存的指令时，它会把内
地址传递给一级数据缓存（或可戏称为L1D$，因为英语中“缓存（cache）”和&ldquo
现金（cash）”的发音相同）。一级数据缓存会检查它是否有这个内存地址对应的缓存段。如
没有，它会把整个缓存段从内存（或者从更高一级的缓存，如果有的话）中加载进来。是的，一次加
整个缓存段，这是基于这样一个假设：内存访问倾向于本地化（localized），如果我们当前需要某个
址的数据，那么很可能我们马上要访问它的邻近地址。一旦缓存段被加载到缓存中，读指令就可以正
进行读取。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon
-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', '
egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo
t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t
xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we
kit-text-stroke-width: 0px;" data-source-line="15">如果我们只处理读操作，那么事情会很简单
因为所有级别的缓存都遵守以下规律，我称之为：
<blockquote style="box-sizing: border-box; margin: 0px 0px 16px; padding: 0px 15px; color:
777777; border-left-width: 4px; border-left-style: solid; border-left-color: #dddddd; font-famil
: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe
I Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weig
t: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-inde
t: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-
troke-width: 0px;" data-source-line="17">
基本定律：在任意
刻，任意级别缓存中的缓存段的内容，等同于它对应的内存中的内容。
</blockquote>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon
-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', '
egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo
t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t
xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we
kit-text-stroke-width: 0px;" data-source-line="19">一旦我们允许写操作，事情就变得复杂一点
。这里有两种基本的写模式：直写（write-through）和回写（write-back）。直写更简单一点：我
透过本级缓存，直接把数据写到下一级缓存（或直接到内存）中，如果对应的段被缓存了，我们同时
新缓存中的内容（甚至直接丢弃），就这么简单。这也遵守前面的定律：缓存中的段永远和它对应的
存内容匹配。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon
-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', '
egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo
t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t
xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we
kit-text-stroke-width: 0px;" data-source-line="21">回写模式就有点复杂了。缓存不会立即把写
作传递到下一级，而是仅修改本级缓存中的数据，并且把对应的缓存段标记为“脏”段
脏段会触发回写，也就是把里面的内容写到对应的内存或下一级缓存中。回写后，脏段又变“
净”了。当一个脏段被丢弃的时候，总是先要进行一次回写。回写所遵循的规律有点不同。</
>
<blockquote style="box-sizing: border-box; margin: 0px 0px 16px; padding: 0px 15px; color:
777777; border-left-width: 4px; border-left-style: solid; border-left-color: #dddddd; font-famil
: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe
I Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weig

t: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-inde t: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text- troke-width: 0px;" data-source-line="23">
回写定律：当所有脏段被回写后，任意级别缓存中的缓存段的内容，等同于它对应的内存中的内容
</blockquote>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="25">换句话说，回写模式的定律中，我们去掉了&l quo;在任意时刻”这个修饰语，代之以弱化一点的条件：要么缓存段的内容和内存一致（如果存段是干净的话），要么缓存段中的内容最终要回写到内存中（对于脏缓存段来说）。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="27">直接模式更简单，但是回写模式有它的优势它能过滤掉对同一地址的反复写操作，并且，如果大多数缓存段都在回写模式下工作，那么系统经常以一下子写一大片内存，而不是分成小块来写，前者的效率更高。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="29">有些（大多数是比较老的）CPU只使用直写式，有些只使用回写模式，还有一些，一级缓存使用直写而二级缓存使用回写。这样做虽然在一级和级缓存之间产生了不必要的数据流量，但二级缓存和更低级缓存或内存之间依然保留了回写的优势。想说的是，这里涉及到一系列的取舍问题，且不同的设计有不同的解决方案。没有人规定各级缓存的小必须一致。举个例子，我们会看到有CPU的一级缓存是32字节，而二级缓存却有128字节。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="31">为了简化问题，我省略了一些内容：缓存关性（cache associativity），缓存组（cache sets），使用分配写（write-allocate）还是非分配写（面我描述的直写是和分配写相结合的，而回写是和非分配写相结合的），非对齐的访问（unaligned a cess），基于虚拟地址的缓存。如果你感兴趣，所有这些内容都可以去查查资料，但我不准备在这里了。
<h2 style="box-sizing: border-box; padding-bottom: 0.3em; border-bottom-width: 1px; bord r-bottom-style: solid; border-bottom-color: #eeeeee; margin-top: 1em; margin-bottom: 16px; ont-weight: bold; line-height: 1.225; font-size: 1.75em; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'S goe UI Symbol'; font-style: normal; font-variant: normal; letter-spacing: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word- pacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="33">一致性协议（Coherency rotocols）</h2>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t

xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="35">只要系统只有一个CPU核在工作，一切都没题。如果有多个核，每个核又都有自己的缓存，那么我们就遇到问题了：如果某个CPU缓存段中对应内存内容被另外一个CPU偷偷改了，会发生什么？</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="37">好吧，答案很简单：什么也不会发生。这很糕。因为如果一个CPU缓存了某块内存，那么在其他CPU修改这块内存的时候，我们希望得到通知。们拥有多组缓存的时候，真的需要它们保持同步。或者说，系统的内存在各个CPU之间无法做到与生来的同步，我们实际上是需要一个大家都能遵守的方法来达到同步的目的。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="39">注意，这个问题的根源是我们拥有多组缓存而不是多个CPU核。我们也可以这样解决问题，让多个CPU核共用一组缓存：也就是说只有一块一级存，所有处理器都必须共用它。在每一个指令周期，只有一个幸运的CPU能通过一级缓存做内存操作运行它的指令。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="41">这本身没问题。唯一的问题就是太慢了，因这下处理器的时间都花在排队等待使用一级缓存了（并且处理器会做大量的这种操作，至少每个读写令都要做一次）。我指出这一点是因为它表明了问题不是由多核引起的，而是由多缓存引起的。我们道了只有一组缓存也能工作，只是太慢了，接下来最好就是能做到：使用多组缓存，但使它们的行为起来就像只有一组缓存那样。缓存一致性协议就是为了做到这一点而设计的。就像名称所暗示的那样这类协议就是要使多组缓存的内容保持一致。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="43">缓存一致性协议有多种，但是你日常处理的多数计算机设备使用的都属于“窥探（snooping）”协议，这也是我这里要讲的。（有一种叫“基于目录的（directory-based）”协议，这种协议的延迟性较大，但是在有很多个处理器的系统中，它有更好的可扩展性。）</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="45">“窥探”背后的基本思想是，有内存传输都发生在一条共享的总线上，而所有的处理器都能看到这条总线：缓存本身是独立的，但内存是共享资源，所有的内存访问都要经过仲裁（arbitrate）：同一个指令周期中，只有一个缓存可读写内存。窥探协议的思想是，缓存不仅仅在做内存传输的时候才和总线打交道，而是不停地在窥探线上发生的数据交换，跟踪其他缓存在做什么。所以当一个缓存代表它所属的处理器去读写内存时，他处理器都会得到通知，它们以此来使自己的缓存保持同步。只要某个处理器一写内存，其他处理器上就知道这块内存在它们自己的缓存中对应的段已经失效。</p>

<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="47">在直写模式下，这是很直接的，因为写操作旦发生，它的效果马上会被“公布”出去。但是如果混着回写模式，就有问题了。因为可能在写指令执行过后很久，数据才会被真正回写到物理内存中——在这段时间内，他处理器的缓存也可能会傻乎乎地去写同一块内存地址，导致冲突。在回写模型中，简单把内存写操的信息广播给其他处理器是不够的，我们需要做的是，在修改本地缓存之前，就要告知其他处理器。懂了细节，就找到了处理回写模式这个问题的最简单方案，我们通常叫做MESI协议（译者注：MESI Modified、Exclusive、Shared、Invalid的首字母缩写，代表四种缓存状态，下面的译文中可能会以个字母指代相应的状态）。</p>
<h2 style="box-sizing: border-box; padding-bottom: 0.3em; border-bottom-width: 1px; bord r-bottom-style: solid; border-bottom-color: #eeeeee; margin-top: 1em; margin-bottom: 16px; ont-weight: bold; line-height: 1.225; font-size: 1.75em; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'S goe UI Symbol'; font-style: normal; font-variant: normal; letter-spacing: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word- pacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="49">MESI以及衍生协议</h2>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="51">本节叫做“MESI以及衍生协议&rdquo ，是因为MESI衍生了一系列紧密相关的一致性协议。我们先从原生的MESI协议开始：MESI是四种缓段状态的首字母缩写，任何多核系统中的缓存段都处于这四种状态之一。我将以相反的顺序逐个讲解因为这个顺序更合理：</p>
<ul style="box-sizing: border-box; padding: 0px 0px 0px 2em; margin-top: 0px; margin-bott m: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sa s-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: no mal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orp ans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="54">
<li style="box-sizing: border-box;">失效（Invalid）缓存段，要么已经不在缓存中，要么它的内已经过时。为了达到缓存的目的，这种状态的段将会被忽略。一旦缓存段被标记为失效，那效果就等于它从来没被加载到缓存中。
<li style="box-sizing: border-box;">共享（Shared）缓存段，它是和主内存内容保持一致的一份贝，在这种状态下的缓存段只能被读取，不能被写入。多组缓存可以同时拥有针对同一内存地址的共缓存段，这就是名称的由来。
<li style="box-sizing: border-box;">独占（Exclusive）缓存段，和S状态一样，也是和主内存内容持一致的一份拷贝。区别在于，如果一个处理器持有了某个E状态的缓存段，那其他处理器就不能同持有它，所以叫“独占”。这意味着，如果其他处理器原本也持有同一缓存段，那么它马上变成“失效”状态。
<li style="box-sizing: border-box;">已修改（Modified）缓存段，属于脏段，它们已经被所属的理器修改了。如果一个段处于已修改状态，那么它在其他处理器缓存中的拷贝马上会变成失效状态，个规律和E状态一样。此外，已修改缓存段如果被丢弃或标记为失效，那么先要把它的内容回写到内中——这和回写模式下常规的脏段处理方式一样。

<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t

xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="59">如果把以上这些状态和单核系统中回写模式缓存做对比，你会发现I、S和M状态已经有对应的概念：失效/未载入、干净以及脏的缓存段。所以这的新知识只有E状态，代表独占式访问。这个状态解决了“在我们开始修改某块内存之前，我需要告诉其他处理器”这一问题：只有当缓存段处于E或M状态时，处理器才能去写它，也就说只有这两种状态下，处理器是独占这个缓存段的。当处理器想写某个缓存段时，如果它没有独占权它必须先发送一条“我要独占权”的请求给总线，这会通知其他处理器，把它们拥有的一缓存段的拷贝失效（如果它们有的话）。只有在获得独占权后，处理器才能开始修改数据—mdash;并且此时，这个处理器知道，这个缓存段只有一份拷贝，在我自己的缓存里，所以不会有任冲突。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; txt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="61">反之，如果有其他处理器想读取这个缓存段我们马上能知道，因为我们一直在窥探总线），独占或已修改的缓存段必须先回到“共享”状态。如果是已修改的缓存段，那么还要先把内容回写到内存中。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; txt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="63">MESI协议是一个合适的状态机，既能处理来本地处理器的请求，也能把信息广播到总线上。我不打算讲更多关于状态图的细节以及不同的状态转类型。如果你感兴趣的话，可以在关于硬件架构的书中找到更多的深度内容，但对于本文来说，讲这东西有点过了。作为一个软件开发者，你只要理解以下两点，就大有可为：
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; txt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="65">第一，在多核系统中，读取某个缓存段，实上会牵涉到和其他处理器的通讯，并且可能导致它们发生内存传输。写某个缓存段需要多个步骤：在写任何东西之前，你首先要获得独占权，以及所请求的缓存段的当前内容的拷贝（所谓的“带限获取的读（Read For Ownership）”请求）。
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; txt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="67">第二，尽管我们为了一致性问题做了额外的作，但是最终结果还是非常有保证的。即它遵守以下定理，我称之为：
<blockquote style="box-sizing: border-box; margin: 0px 0px 16px; padding: 0px 15px; color: 777777; border-left-width: 4px; border-left-style: solid; border-left-color: #dddddd; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="69">
MESI定律：在所有脏缓存段（M状态）被回写后，任意缓存级别的所有缓存段中的内容，和它们对应的内存中的内容一。此外，在任意时刻，当某个位置的内存被一个处理器加载入独占缓存段时（E状态），那它就不会出现在其他任何处理器的缓存中。

</blockquote>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="71">注意，这其实就是我们已经讲过的回写定律 上独占规则而已。我认为MESI协议或多核系统的存在根本没有弱化我们现有的内存模型。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="73">好了，至此我们（粗略）讲了原生MESI协议 以及使用它的CPU，比如ARM）。其他处理器使用MESI扩展后的变种。常见的扩展包括“O&r quo;（Owned）状态，它和E状态类似，也是保证缓存间一致性的手段，但它直接共享脏段的内容， 不需要先把它们回写到内存中（“脏段共享”），由此产生了MOSEI协议。还有MERSI MESIF，这两个名字代表同一种思想，即指定某个处理器专门处理针对某个缓存段的读操作。当多个 理器同时拥有某个S状态的缓存段的时候，只有被指定的那个处理器（对应的缓存段为R或F状态）才 对读操作做出回应，而不是每个处理器都能这么做。这种设计可以降低总线的数据流量。当然你可以 时加入R/F状态和O状态，或者更多的状态。这些都属于优化，没有一种会改变基本定律，也没有一 会改变MESI协议所确保的结果。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="75">我不是这方面的专家，很有可能有系统在使 其他协议，这些协议并不能完全保证一致性，不过如果有，我没有注意到它们，或者没有看到有什么 行的处理器在使用它们。所以为了达到我们的目的，我们真的就可以假设一致性协议能保证缓存的一 性。不是基本一致，不是“写入一会儿后才能保持一致”——而是完全 一致。从这个层面上说，除非硬件有问题，内存的状态总是一致的。用技术术语来说，MESI以及它 衍生协议，至少在原理上，提供了完整的顺序一致性（sequential consistency），在C++ 11的内存 型中，这是最强的一种确保内存顺序的模型。这也引出了问题，为什么我们需要弱一点的内存模型， 及“什么时候会用到它们”？</p>
<h2 style="box-sizing: border-box; padding-bottom: 0.3em; border-bottom-width: 1px; bord r-bottom-style: solid; border-bottom-color: #eeeeee; margin-top: 1em; margin-bottom: 16px; ont-weight: bold; line-height: 1.225; font-size: 1.75em; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'S goe UI Symbol'; font-style: normal; font-variant: normal; letter-spacing: normal; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word- pacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="77">内存模型</h2>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="79">不同的体系结构提供不同的内存模型。到本 写作的时候为止，ARM和POWER体系结构的机器拥有相对较弱的内存模型：这类CPU在读写指令重 序（reordering）方面有相当大的自由度，这种重排序有可能会改变程序在多核环境下的语义。通过& dquo;内存屏障（memory barrier）”，程序可以对此加以限制：“重排序操作不允许 过这条边界”。相反，x86则拥有较强的内存模型。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', '

egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="81">我不打算在这里深入到内存模型的细节中，很容易陷入堆砌技术术语中，而且也超出了本文的范围。但是我想说一点关于“他们如何发生&dquo;的内容——也就是，弱内存模型如何保证正确性（相比较于MESI协议给缓存带的顺序一致性），以及为什么。当然，一切都归结于性能。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="83">规则是这样的：如果满足下面的条件，你就以得到完全的顺序一致性：第一，缓存一收到总线事件，就可以在当前指令周期中迅速做出响应。第，处理器如实地按程序的顺序，把内存操作指令送到缓存，并且等前一条执行完后才能发送下一条。然，实际上现代处理器一般都无法满足以上条件：</p>
<ul style="box-sizing: border-box; padding: 0px 0px 0px 2em; margin-top: 0px; margin-bott m: 16px; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sa s-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: no mal; font-variant: normal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orp ans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="85">
<li style="box-sizing: border-box;">缓存不会及时响应总线事件。如果总线上发来一条消息，要使个缓存段失效，但是如果此时缓存正在处理其他事情（比如和CPU传输数据），那这个消息可能无法当前的指令周期中得到处理，而会进入所谓的“失效队列（invalidation queue）”，个消息等在队列中直到缓存有空为止。
<li style="box-sizing: border-box;">处理器一般不会严格按照程序的顺序向缓存发送内存操作指令当然，有乱序执行（Out-of-Order execution）功能的处理器肯定是这样的。顺序执行（in-order ex cution）的处理器有时候也无法完全保证内存操作的顺序（比如想要的内存不在缓存中时，CPU就不为了载入缓存而停止工作）。
<li style="box-sizing: border-box;">写操作尤其特殊，因为它分为两阶段操作：在写之前我们先要到缓存段的独占权。如果我们当前没有独占权，我们先要和其他处理器协商，这也需要一些时间。同，在这种场景下让处理器闲着无所事事是一种资源浪费。实际上，写操作首先发起获得独占权的请求然后就进入所谓的由“写缓冲（store buffer）”组成的队列（有些地方使用“缓冲”指代整个队列，我这里使用它指代队列的一条入口）。写操作在队列中等待，直到缓存备好处理它，此时写缓冲就被“清空（drained）”了，缓冲区被回收用于处理新的写作。

<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="89">这些特性意味着，默认情况下，读操作有可会读到过时的数据（如果对应失效请求还等在队列中没执行），写操作真正完成的时间有可能比它们代码中的位置晚，一旦牵涉到乱序执行，一切都变得模棱两可。回到内存模型，本质上只有两大阵营</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 16px; color: #333333; fon -family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', ' egoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: normal; fo t-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; t xt-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -we kit-text-stroke-width: 0px;" data-source-line="91">在弱内存模型的体系结构中，处理器为了开者能写出正确的代码而做的工作是最小化的，指令重排序和各种缓冲的步骤都是被正式允许的，也就

说没有任何保证。如果你需要确保某种结果，你需要自己插入合适的内存屏障——它防止重排序，并且等待队列中的操作全部完成。</p>
<p style="box-sizing: border-box; margin-top: 0px; margin-bottom: 0px !important; color: #333333; font-family: 'Helvetica Neue', Helvetica, 'Segoe UI', Arial, freesans, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol'; font-size: 16px; font-style: normal; font-variant: ormal; font-weight: normal; letter-spacing: normal; line-height: 25.6px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px;" data-source-line="93">使用强一点的内存模型的体系结构会在内部做很多记录工作。比如，x86会跟踪所有在等待中的内存操作，这些操作都还没有完全完成称为“退休（retired）”）。它会把它们的信息保存在芯片内部的MOB（“memory ordering buffer”，内存排序缓冲）。x86作为部分支持乱序执行的体系结构，在出问题时候能把尚未“退休”的指令撤销掉——比如发生页错误（page fault，或者分支预测失败（branch mispredict）的时候。我已经在我以前的文章“好奇地说”中提到过一些细节，以及和内存子系统的一些交互。主旨是x86处理器会主动地监控外部事件（比如存失效），有些已经执行完的操作会因为这些事件而被撤销，但不算“退休”。这就是，x86知道自己的内存模型应该是什么样子的，当发生了一件和这个模型冲突的事，处理器会回退到一个与内存模型兼容的状态。这就是我在以前另一篇文章中提到的“清除内存排序机（memory ordering machine clear）”。最后的结果是，x86处理器为内存操作提供了很强的一致性保——虽然没有达到完美的顺序一致性。</p>