*By William Zola, Lead Technical Support Engineer at MongoDB*

在上一篇文章中我介绍了三种基本的设计方案：内嵌，子引用，父引用，同时说明了在选择方案时需要考虑的两个关键因素。

一对多中的多是否需要一个单独的实体。

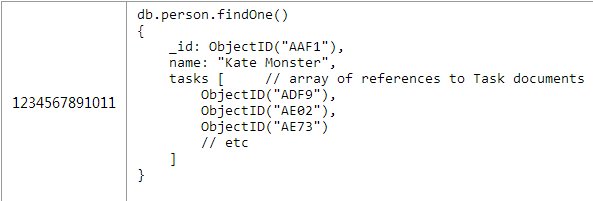
这个关系中集合的规模是一对很少，很多，还是非常多。

在掌握了以上基础技术后，我将会介绍更为高级的主题：双向关联和反范式化。

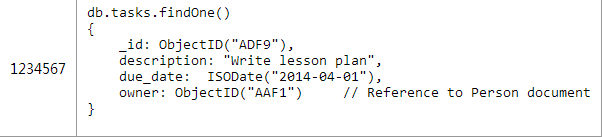
**双向关联**

如果你想让你的设计更酷，你可以让引用的“one”端和“many”端同时保存对方的引用。

以上一篇文章讨论过的任务跟踪系统为例。有person和task两个集合，one-to-n的关系是从person端到task端。在需要获取person所有的task这个场景下需要在person这个对象中保存有task的id数组，如下面代码所示。



在某些场景中这个应用需要显示任务的列表（例如显示一个多人协作项目中所有的任务），为了能够快速的获取某个用户负责的项目可以在task对象中嵌入附加的person引用关系。



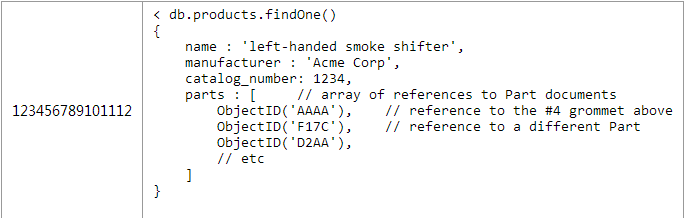
这个方案具有所有的一对多方案的优缺点，但是通过添加附加的引用关系。在task文档对象中添加额外的“owner”引用可以很快的找到某个task的所有者，但是如果想将一个task分配给其他person就需要更新引用中的person和task这两个对象（熟悉关系数据库的童鞋会发现这样就没法保证操作的原子性。当然，这对任务跟踪系统来说并没有什么问题，但是你必须考虑你的用例是否能够容忍）

**在一对多关系中应用反范式**

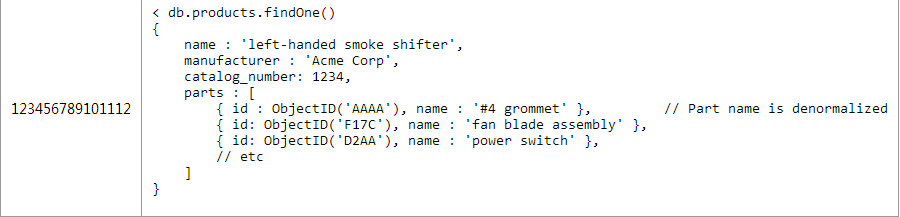
在你的设计中加入反范式，可以使你避免应用层级别的join读取，当然，代价是这也会让你在更新是需要操作更多数据。下面我会举个例子来进行说明

**反范式Many -< One**

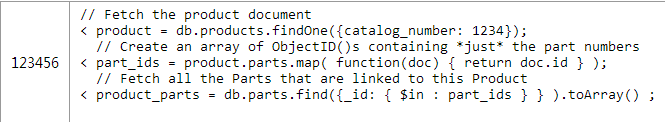
以产品和零件为例，你可以在parts数组中冗余存储零件的名字。以下是没有加入反范式设计的结构。



反范式化意味着你不需要执行一个应用层级别的join去显示一个产品所有的零件名字，当然如果你同时还需要其他零件信息那这个应用层的join是避免不了的。



在使得获取零件名字简单的同时，执行一个应用层级别的join会和之前的代码有些区别，具体如下：



反范式化在节省你读的代价的同时会带来更新的代价：如果你将零件的名字冗余到产品的文档对象中，那么你想更改某个零件的名字你就必须同时更新所有包含这个零件的产品对象。

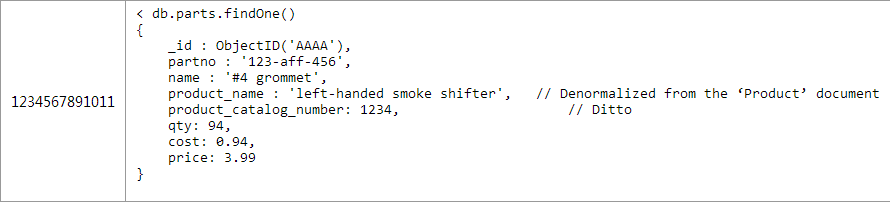
在一个读比写频率高的多的系统里，反范式是有使用的意义的。如果你很经常的需要高效的读取冗余的数据，但是几乎不去变更他d话，那么付出更新上的代价还是值得的。更新的频率越高，这种设计方案的带来的好处越少。

例如：假设零件的名字变化的频率很低，但是零件的库存变化很频繁，那么你可以冗余零件的名字到产品对象中，但是别冗余零件的库存。

需要注意的是，一旦你冗余了一个字段，那么对于这个字段的更新将不在是原子的。和上面双向引用的例子一样，如果你在零件对象中更新了零件的名字，那么更新产品对象中保存的名字字段前将会存在短时间的不一致。

**反范式One -< Many**

你也可以冗余one端的数据到many端：

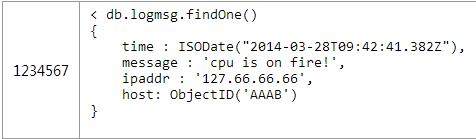


如果你冗余产品的名字到零件表中，那么一旦更新产品的名字就必须更新所有和这个产品有关的零件，这比起只更新一个产品对象来说代价明显更大。这种情况下，更应该慎重的考虑读写频率。

**在一对很多的关系中应用反范式**

在日志系统这个一对许多的例子中也可以应用反范式化的技术。你可以将one端（主机对象）冗余到日志对象中，或者反之。

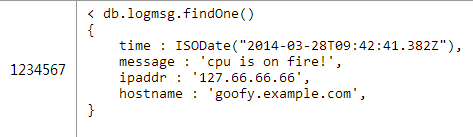
下面的例子将主机中的IP地址冗余到日志对象中。



如果想获取最近某个ip地址的日志信息就变的很简单，只需要一条语句而不是之前的两条就能完成。

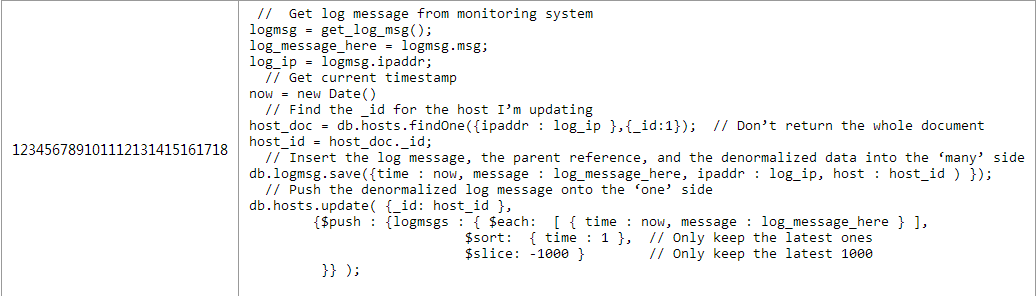
http://upload-images.jianshu.io/upload_images/14570-734de07cf508d8c3.png

事实上，如果one端只有少量的信息存储，你甚至可以全部冗余存储到多端上，合并两个对象。



另一方面，也可以冗余数据到one端。比如说你想在主机文档中保存最近的1000条日志，可以使用mongodb 2.4中新加入的$eache/$slice功能来保证list有序而且只保存1000条。

日志对象保存在logmsg集合中，同时冗余到hosts对象中。这样即使hosts对象中超过1000条的数据也不会导致日志对象丢失。



通过在查询中使用投影参数 （类似{\_id:1}）的方式在不需要使用logmsgs数组的情况下避免获取整个mongodb对象，1000个日志信息带来的网络开销是很大的。

在一对多的情况下，需要慎重的考虑读和更新的频率。冗余日志信息到主机文档对象中只有在日志对象几乎不会发生更新的情况下才是个好的决定。

**总结**

在这篇文章里，我介绍了对三种基础方案：内嵌文档，子引用，父引用的补充选择。

使用双向引用来优化你的数据库架构，前提是你能接受无法原子更新的代价。

可以在引用关系中冗余数据到one端或者N端。

在决定是否采用反范式化时需要考虑下面的因素：

**你将无法对冗余的数据进行原子更新。**

**只有读写比较高的情况下才应该采取反范式化的设计。**