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

“我有丰富的sql使用经验，但是我是个MongoDB的初学者。我应该如何在MongoDB中针对一对多关系进行建模？”这是我被问及最多的问题之一。

我没法简单的给出答案，因为这有很多方案去实现。接下来我会教导你如何针对一对多进行建模。

这个话题有很多内容需要讨论，我会用三个部分进行说明。在第一部分，我会讨论针对一对多关系建模的三种基础方案。在第二部分我将会覆盖更多高级内容，包括反范式化和双向引用。在最后一部分，我将会回顾各种选择，并给出做决定时需要考虑的因素。

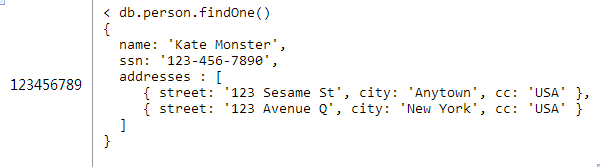
很多初学者认为在MongoDB中针对一对多建模唯一的方案就是在父文档中内嵌一个数组子文档，但是这是不准确的。因为你可以在MongoDB内嵌一个文档不代表你就必须这么做。

当你设计一个MongoDB数据库结构，你需要先问自己一个在使用关系型数据库时不会考虑的问题：这个关系中集合的大小是什么样的规模？你需要意识到一对很少，一对许多，一对非常多，这些细微的区别。不同的情况下你的建模也将不同。

*Basics: Modeling One-to-Few*

一对很少

针对个人需要保存多个地址进行建模的场景下使用内嵌文档是很合适，可以在person文档中嵌入addresses数组文档：



这种设计具有内嵌文档设计中所有的优缺点。最主要的优点就是不需要单独执行一条语句去获取内嵌的内容。最主要的缺点是你无法把这些内嵌文档当做单独的实体去访问。

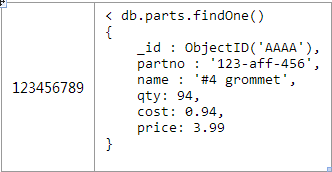
例如，如果你是在对一个任务跟踪系统进行建模，每个用户将会被分配若干个任务。内嵌这些任务到用户文档在遇到“查询昨天所有的任务”这样的问题时将会非常困难。我会在下一篇文章针对这个用例提供一些适当的设计。

*Basics: One-to-Many*

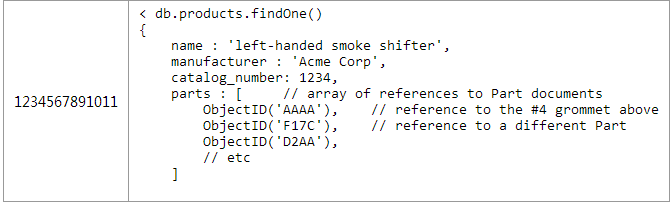
一对许多

以产品零件订货系统为例。每个商品有数百个可替换的零件，但是不会超过数千个。这个用例很适合使用间接引用---将零件的objectid作为数组存放在商品文档中(在这个例子中的ObjectID我使用更加易读的2字节，现实世界中他们可能是由12个字节组成的)。

每个零件都将有他们自己的文档对象



每个产品的文档对象中parts数组中将会存放多个零件的ObjectID ：



在获取特定产品中所有零件，需要一个应用层级别的join

为了能快速的执行查询，必须确保products.catalog\_number有索引。当然由于零件中parts.\_id一定是有索引的，所以这也会很高效。

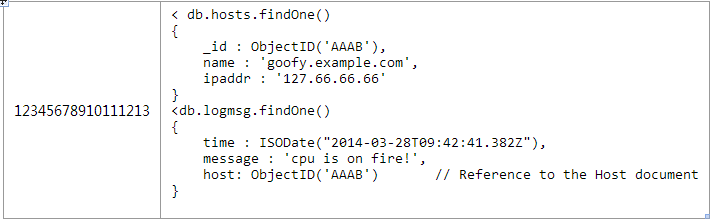
这种引用的方式是对内嵌优缺点的补充。每个零件是个单独的文档，可以很容易的独立去搜索和更新他们。需要一条单独的语句去获取零件的具体内容是使用这种建模方式需要考虑的一个问题（请仔细思考这个问题，在第二章反反范式化中，我们还会讨论这个问题）

这种建模方式中的零件部分可以被多个产品使用，所以在多对多时不需要一张单独的连接表。

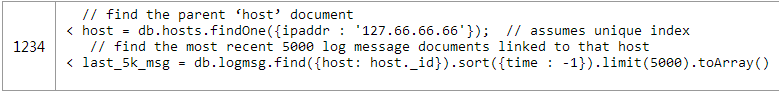
*Basics: One-to-Squillions*

一对非常多

我们用一个收集各种机器日志的例子来讨论一对非常多的问题。由于每个mongodb的文档有16M的大小限制，所以即使你是存储ObjectID也是不够的。我们可以使用很经典的处理方法“父级引用”---用一个文档存储主机，在每个日志文档中保存这个主机的ObjectID。



以下是个和第二中方案稍微不同的应用级别的join用来查找一台主机最近5000条的日志信息



所以，即使这种简单的讨论也有能察觉出mongobd的建模和关系模型建模的不同之处。你必须要注意一下两个因素：

*Will the entities on the “N” side of the One-to-N ever need to stand alone?*

一对多中的多是否需要一个单独的实体。

*What is the cardinality of the relationship: is it one-to-few; one-to-many; or one-to-squillions?*

这个关系中集合的规模是一对很少，很多，还是非常多。

*Based on these factors, you can pick one of the three basic One-to-N schema designs:*

基于以上因素来决定采取一下三种建模的方式

一对很少且不需要单独访问内嵌内容的情况下可以使用内嵌多的一方。

一对多且多的一端内容因为各种理由需要单独存在的情况下可以通过数组的方式引用多的一方的。

一对非常多的情况下，请将一的那端引用嵌入进多的一端对象中。

下一次我们将会看到如何使用双向关系和反范式化去提升以上三种基本方案的性能。