# 为什么你不应该使用 MongoDB

免责声明：我不构建数据库引擎，但搭建Web应用。每年我大约跑4-6个不同项目，所以我搭建了不少Web应用。我经历过有不同需求及不同的数据存储需求的应用。我部署过你听说过或没听说过的的大部分数据存储。

我也有几次做出了错误的选择。这是一个关于其中一次的故事——原本我们为什么选择它，我们是如何发现它不合适，以及我们如何修复的。这一切都发生在一个开放源码的名为Diaspora的项目中。

### Diaspora项目

[Diaspora](http://www.oschina.net/p/diaspora) 是一个有着悠久历史的分布式社交网络。早在2010年初，纽约大学四名大学生创建了一个Kickstarter视频，目的是募集10000美元，耗费一个夏天来构建一个替代Facebook的分布式项目。他们将它发送给朋友和家人，并希望有最好的结果。

但他们触到了痛处。当时正好有另一起关于Facebook的隐私丑闻，当尘埃落定，回到他们的Kickstarter项目时，他们已经从6400个不同的人那里筹集了超过200000美元，而同时这个软件项目，却连一行代码都还没有写出来。

Diaspora是第一个大大超越其目标的Kickstarter项目。其结局是，他们的项目新闻被刊登于纽约时报——但它变成了一个丑闻，因为团队照片 的背景黑板上面有一个肮脏的笑话，直到最终印刷都没有人注意到这一点。这可是刊登在纽约时报！这个事件的余波实际上是我第一次听说到这个项目。

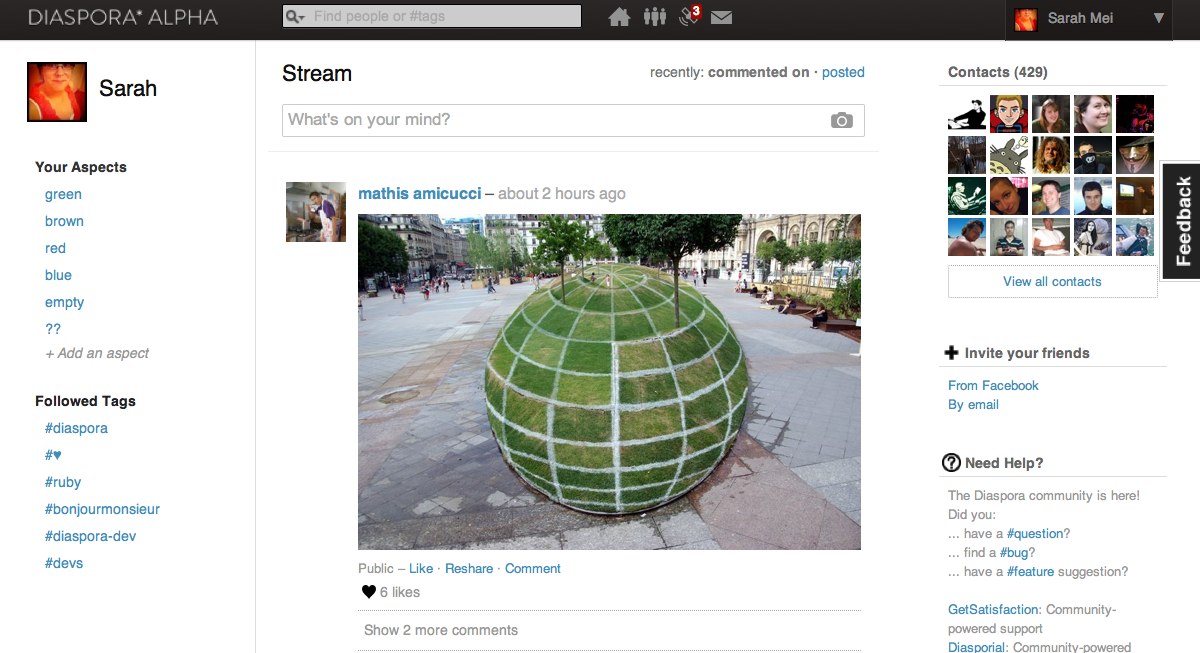
他们在Kickstarter上成功的结果，是离开学校，来到San Francisco开始写代码。他们最后在我的办公室工作。那时我在Pivotal实验室工作，他们的一个哥哥也在那工作，所以Pivotal给他们提供 了免费的办公空间、网络，当然，装有啤酒的冰箱。我白天和官方客户工作，下班后和他们一起玩，周末贡献代码。

他们最后在Pivotal待了两年多的时间。虽然只是在第一个暑假的结尾，他们已经有了一个最小化的，但已经可以使用（在某种意义上说）的一个分布式社交网络，以Ruby on Rails实现，后端用MongoDB。

这里有很多流行词汇，让我们把它分解来看。

### “分布式社交网络”

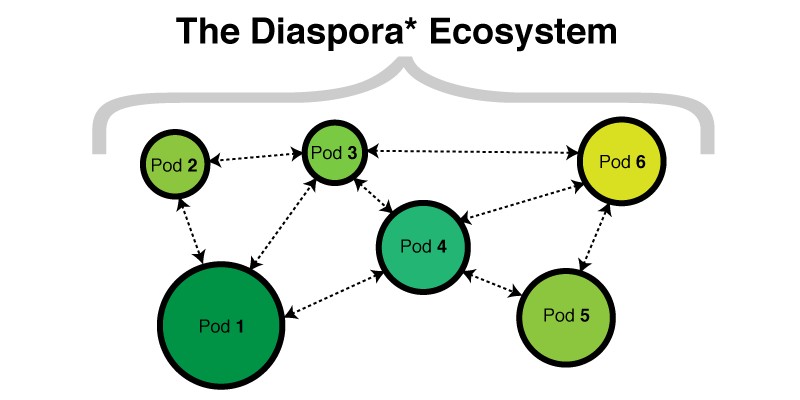
如果你看过《社交网络》，你就知道了所有你需要知道的关于Facebook的事。它是个web应用，它在一个单逻辑服务器上运行，它可以让你与朋友保持联系。一旦你登录，Diaspora的界面在结构上与Facebook很相似：



Diaspora用户界面的截图

在中间有个信息流，显示你所有朋友发布的信息，在边上有一些其他随机的基本没人关注过的东西。Diaspora和Facebook主要的技术差异在于对终端用户的不可见性：它是“分布的”部分。

Diaspora的基础设施并不在一个单独的网络地址上。有几百个独立的Diaspora服务器。代码是开源的，所以如果你愿意，你可以建立自己的服务 器。各个服务器，被称作一个pod，有它自己的数据库和自己的用户集合，并可以与其他拥有各自数据库和用户集的Diaspora pod进行交互操作。



不同规模的Pod互相交流，而没有一个中央枢纽。

每个pod通过基于HTTP的API与其它pod通信。一旦你在一个pod上创建了一个帐号，它其实相当无聊，除非你关注一些其他人。你可以在你的pod 上关注其他用户，而且你也可以关注其它pod上的用户。当你关注的人在另一个pod上发布一个更新时，将会发生这些事情：

1. 该更新会进入作者的pod的数据库。
2. 你的pod会通过API收到通知。
3. 该更新被存入你的pod的数据库。
4. 你可以看看近期活动，将会发现这个发布与你关注的其他人的发布混在一起。

评论按同样的方式运作。在任何单一的发布中，一些评论的评论人可能来自与发布作者相同的pod，而还有一些可能来自其他的pod。任何有权查看这个发布的人将会看到所有的评论，就像你期待的，好像每个人都在一个单一的逻辑服务器上一样。

### 谁关心？

这个架构有技术上和法律上的优势。技术上的主要优势在于容错机制



这是每个办公室都应该有的一个重要的容错系统。

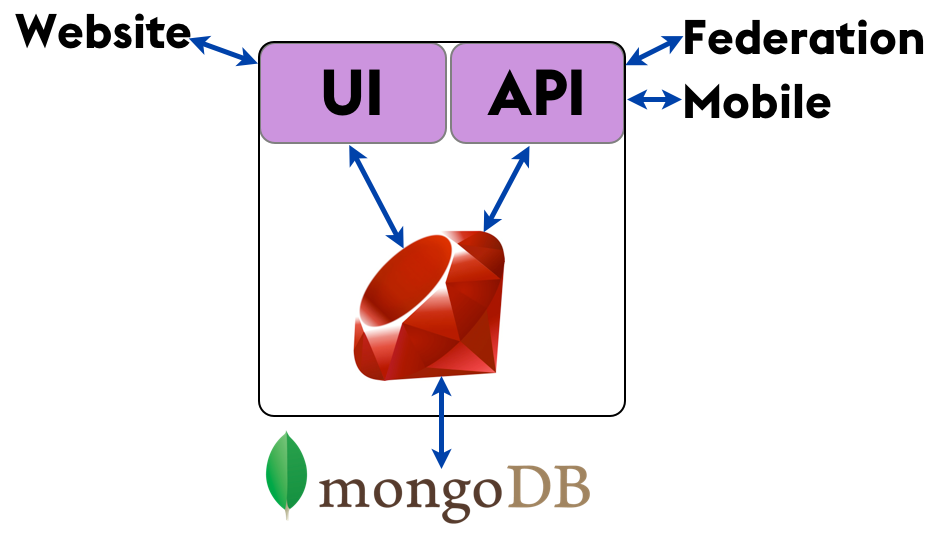
如果pod中的任意一个宕机，这不会引起其他的也宕机。系统的生存，甚至期望，网络的分割。对于这，有一些有趣的政治上的暗示——例如，如果你在一个关闭 了外接网络而无法访问Facebook和Twitter的国家，你的pod依旧会在本地运行，并和你所在国家内的其他人相连接，即使无法访问外部。

主要的法律优势是服务器的独立性。每个pod都是一个法律独立的实体，由其创建所在地的法律所掌管。每个pod也设置自己的服务条款。对于其中的大多数， 你可以在不放弃对内容权限的条件下发布内容，这与Facebook不同。Diaspora是一个免费的软件，其意义在于“免费”和“自由”，而且大多数运 行pod的人非常在意这种事情。

所以这就是这个系统的架构。我们再来从一个单一的pod看看这个架构。

### 这是一个 Rails 应用

每个pod都是一个Ruby on Rails应用，后端有一个数据库，最初是MongoDB。从某种程度来说，这个代码库是一个‘典型’的Rails应用——它同时具有一个可视化与可编程的UI，一些Ruby代码，以及一个数据库。但在其他方面，它决不是典型的。



一个 Diaspora pod的内部结构

视觉UI显然就是网站用户与Diaspora交互的方式。这个API被各种Diaspora移动客户端使用——这一部分相当典型——但同时它也被用 于"federation"（联邦），这也是描述pod之间通讯的技术名词。（我有一次问过这与罗慕伦人的相似点在哪里，结果得到一堆白眼，擦。）所以说 这个系统的分布式特性，给代码库增加了典型应用中所不存在的中间层。

当然，MongoDB是一种数据存储的典型选择。而绝大多数的Rails应用程序是由PostgreSQL（近来不常见）或MySQL所支持。

所以代码部分就是这样。我们再来考虑一下我们存储的是什么样的数据。

（译注：罗慕伦帝国是科幻系列《[星际旅行](http://baike.baidu.com/view/280028.htm" \t "_blank)》中虚构的外星帝国，2158年，罗慕伦帝国与当时的地球联邦发生了一场核战争。这场战争同时威胁到了其他的一些种族，并最终促使了星际联邦的建立。2160年罗慕伦人被打败，后与星际联邦签订合约并划定中立区。）

### 我不认为单词的意思是你认为的意思

“社交数据”是关于我们朋友、他们朋友和他们活动的网络信息。从概念上来看，我们确实认为它是一个网络——一个以我们为中心，朋友围绕在我们身边的无向网络。



所有照片来自rubyfriends.com。感谢Matt Rogers、Steve Klabnik、Nell Shamrell、Katrina Owen、Sam Livingston-Grey、Josh Susser、Akshay Khole、Pradyumna Dandwate和Hephzibah Watharkar对#rubyfriends的贡献！

当我们存储社交数据时，我们存储的是那个图的拓扑结构，和那些随着边移动的活动信息。

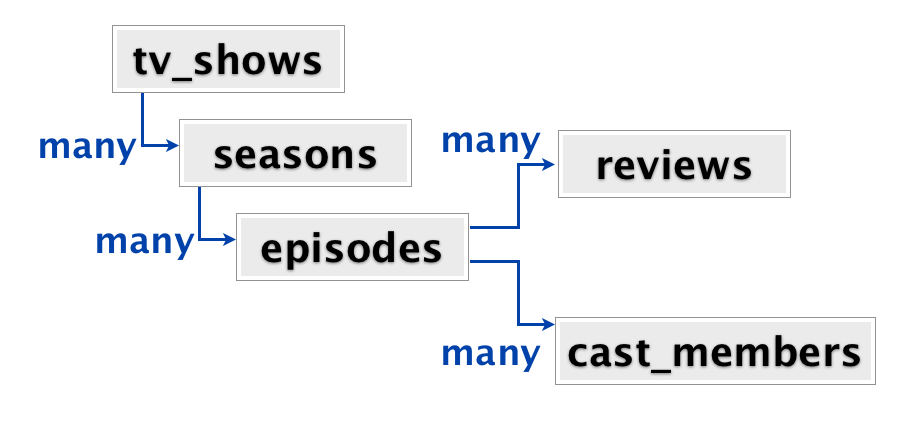
多年之后的今天，大家公认**社交数据并不是关系型的**，如果你把它存储在关系型数据库中，你就错了。

但是，其它方案有哪些呢? 有人说图数据库更自然些，不过这里我不打算介绍它，因为图数据库太过小众而不适合用于生产环境。另外一些人说文档数据库对社交数据来说堪称完美，而且也更为主流化足以投入实际使用。下面，让我们看看为什么人们认为社交数据更适合放在MongoDB里，而不是放在PostgreSQL里。

### MongoDB 如何存储数据

MongoDB是一个面向文档的数据库。它把你的数据存储在由独立的**文档**组成的**集合**中，而不是像关系型数据库那样，存储在由独立的**行**组成的**表**中。在MongoDB中, 一个文档是一大块JSON数据，没有特定的格式或模式。

比如说，你需要对下列一组关系进行建模。这和来自于Pivotal的一个使用了MongoDB的项目类似，是我见过的最适合于文档数据库的用例。

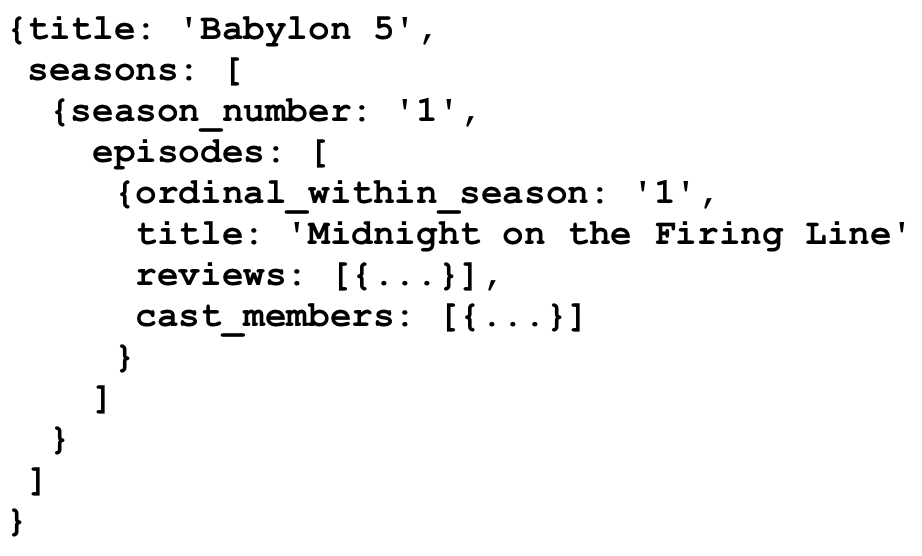


根元素是电视节目。每个节目有很多季，每一季都有很多片段，每个片段都有很多评论和演员表。当用户进入这个网站后，一般都是直接访问一个特定电视节目的页 面。在这个页面里他们可以看到所有的季、所有的片段、所有的评论和所有的演员表。从应用的角度来看，当用户访问一个页面时，我们就将检索**所有**有关电视节目的信息。

有很多方法可以为此数据建模。在典型的关系型数据存储中，上面的每一个方框就是一个表。你必须有一个叫电视节目的表、一个有外键是电视节目的每一季表、一 个有外键是每一季的片段表以及外键是片段的评论表和演员表。所以，要得到电视节目的所有信息，你必须要在5个表中查询。

我们也可以以这样的数据作为一组嵌套的哈希值进行建模。有关特定电视节目的信息的集合是一个大的嵌套的keyvalue数据结构。 在电视节目里有一个季 节的数组，每一个季节是一个hash。 在每个季节里,每一个episodes都是一个hash等等. 这就是Mongo如何建立数据模型的. 每一个电视节目是一个包含我们需要的所有信息的文档。

这是一个电视节目文档的例子, Babylon 5.



它有一些标题的元数据，然后是一个季节的数组. 每个季节本身就是一个带有元数据的哈希数组episodes. 反过来, 每个episode 都有一些reviews和cast\_members的元数据和数组。

这就像是一个巨大的分形体数据结构。



集合的集合的集合的集合。就像是个分形体！

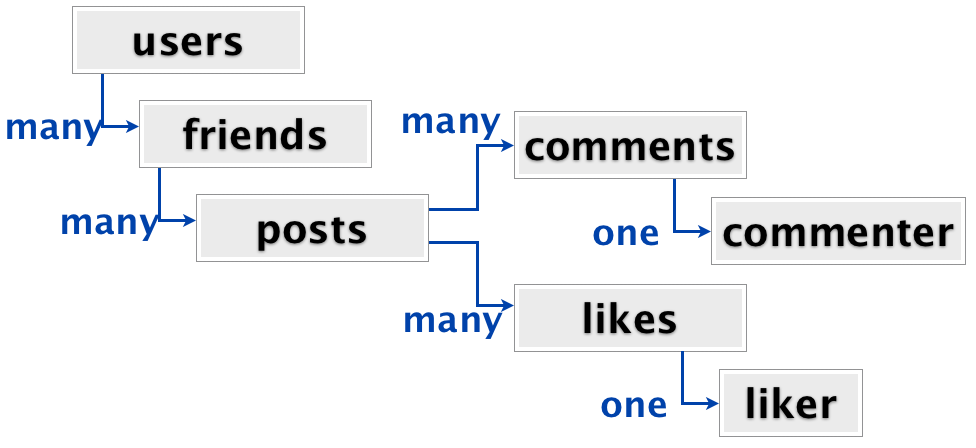
所有我们需要的电视节目数据都在一个文档里，所以要检索一次所有信息时是很快的，即使这个文档很庞大。美剧“综合医院”发布了50+季，超过12000 集。在我的笔记本电脑上，PostgresSQL查询所有数据得1分钟，而在MongoDB中用一个ID查询时是秒级。

所以，不管怎么说，这个应用对于存储文档模型还是不错的选择。

## 好吧，那社交数据又是怎样的？

好的，当你进入社交网站后，映入眼帘的唯一重要页面部分是你的活动流。活动流显示了你所有关注人的信息，这些信息是按最新时间排序的。每一条信息里都是网状结构，比如说图片，喜欢，分享以及评论。

网状结构的活动流看起来和上面提到的电视节目很相似。



用户有朋友，朋友有帖子，帖子有评论和喜欢，每一个评论有一个评论者，每一个喜欢有一个喜欢的人。这种关系并不比电视节目的复杂。和电视节目一样，当用户 登录后，我们就想一次取出所有的数据。此外，在关系数据库中，所有的数据都是规格化的，这就得在7个表中查询才能得到所有数据。

7个表联合查询。啊！如果将每个用户的活动流作为一个大的非标准化的网状结构来存储的话，要比每一次连接查询看起来要好的多。

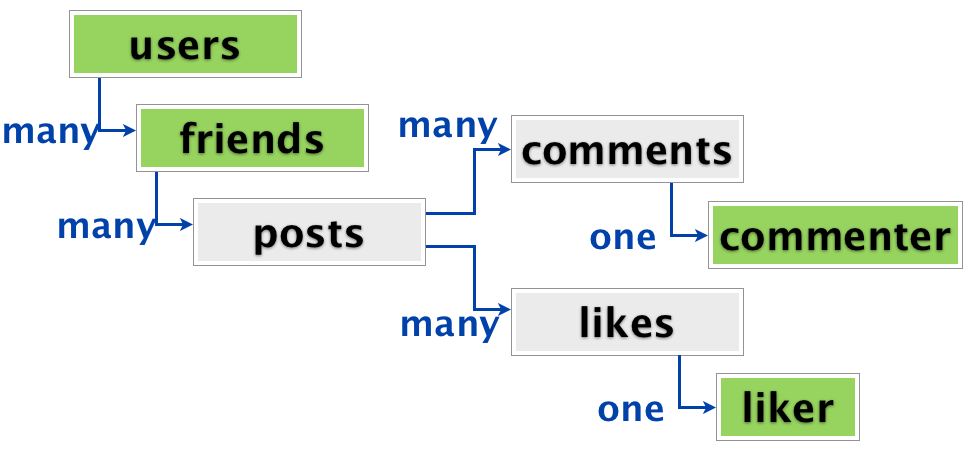
在2010年时，Diaspora团队做出了这个决定，并深受Etsy有关用文档结构存储文章的影响，尽管当时他们曾公开远离了MongoDB数据存储。 同样的，Facebook的Cassandra也曾呼唤要远离关系数据库。Diaspora与时俱进的选择了MongoDB。从他们的信息数据来看，这样 的选择是明智的。

### 问题可能出在哪？

Diaspora的“社交数据”和TV show的Mongo风格数据之间有一个很重要的不同点，我们在开始时都没有注意到。

TV Show中，数据关系表的每一部分都属于不同的数据类型。TV Show，Seasons,Episodes,Reviews,Cast members,他们各不相同。

但对于“社交数据”，有一部分具有相同的数据类型。事实上，图表中所有绿色部分所表示的都是同一种数据类型----Diaspora users



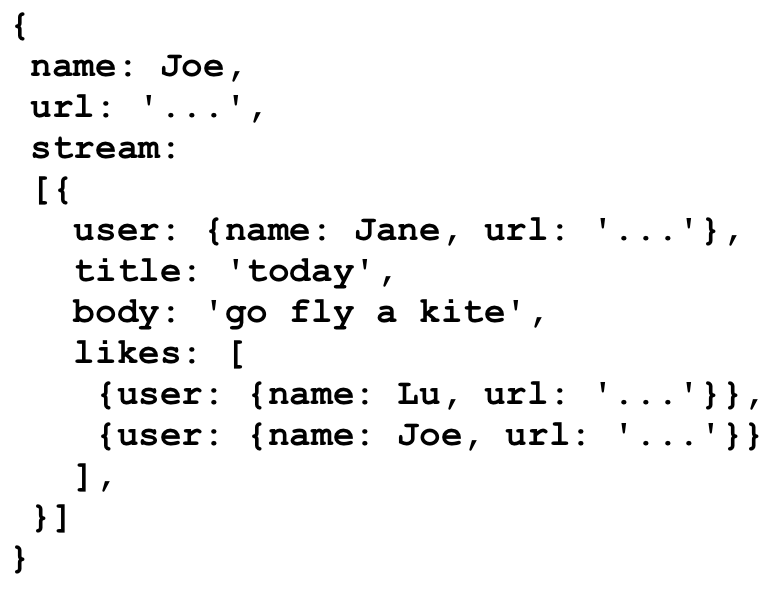
每个user具有一些friends，而每个friend可能自身就是一个user。或者说，他们也可能不是，因为这是一个分布式的系统。（我今天只是跳过了那整个层面的复杂性。）同样的，commenters和likers也可能是users。

这种类型的重复性使得要想将活动流正规化到一个单独的文档变得更难。那是因为在你的文档的不同部分，可能引用了相同的概念——在这个例子中，就是相同的 user。在活动流中喜欢(like)那个帖子(post)的user，可能也是评论(comment)另一个不同帖子(post)的user。

### 重复数据重复数据

在MongoDB中我们可以用几种不同的方式来表示它。复制是一种简单的选择。在第一次提交的时候，friend的所有信息都被复制下来并被保存到 like，之后在第二次提交的时候，一个单独的副本被保存到comment。这里的好处在于，在你需要数据的任何地方，它都是存在的，而且你仍然可以把整 个活动流作为一个单独的文档处理。

这里就是这类完全非规范化的流程文档的样子。



这里有内联的user数据的拷贝。这个是Joe的数据流，而且在最顶级有他的用户数据，包括他的name和URL。紧接着下来是他的数据流，包含有 Jane的帖子(post)。Joe喜欢(like)Jane的帖子，所以在Jane的帖子的喜欢者(likes)中，我们有Joe的数据的一个单独的拷 贝。

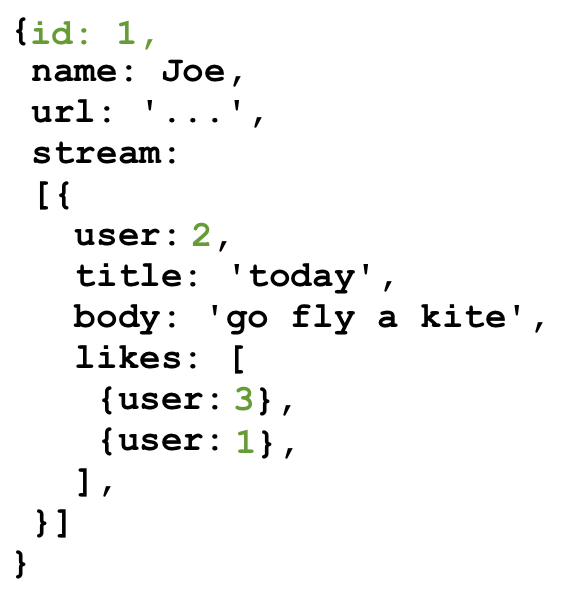
你会明白为什么这样做很有吸引力：所有你需要的数据在你需要的地方已经存在。

你也会同样明白为什么这么做是危险的。更新一个user的数据，就意味着要查找所有他们出现过的活动流，以便在这些不同的地方更新这个数据。这很容易出错，经常导致不一致的数据和奇怪的错误，特别是在处理删除操作的时候。

### 就没有希望了吗?

在MongoDB中你可以采用另一种方法来解决这个问题，如果你有相关背景经验的话会对此更熟悉。与复制用户数据不同的是，你可以在活动流文档中保存用户的引用。

用这个方法，代替那种在需要的地方将用户数据内联进来的方法，你只需给每个user一个ID。一旦用户具有了ID，我们就只需在之前内联数据的地方保存这个用户的ID。下面的这些新的ID是用绿色标识的。



MongoDB实际用的是BSON IDs（译注：BSON是由10gen开发的一个数据格式，目前主要用于MongoDB中，是MongoDB的数据存储格式），它就像是GUID的字符串形式， 但为了让这些例子便于阅读，这里我只是用了整数。

这消除了我们之前的重复的问题。当用户数据改变时，只有一个文档需要重写。不过我们为我们自己带来了一个新的问题。因为我们将一些数据移到了活动流之外， 我们再也不能从一个单独的文档构造一个活动流了。这使得效率降低，复杂性增加。现在构造一个活动流需要 1)检索数据流文档，然后 2)检索所有用户文档以便填写名字和头像。

MongoDB缺少的是SQL风格的join操作，这种操作可以通过写一条查询语句，得到活动流与活动流所引用的所有用户的混合结果。因为MongoDB不具备这种能力，取而代之的是，你必须在你的应用代码中手工做这项混合工作。

### 简单的非规范化数据

我们回头看一看电视节目（TV shows），电视节目的关系集合没有太多的复杂性。因为关系图中所有的盒子是不同的实体，整个查询可以整合进一个文档，没有重复没有引用。在这个文档数据库中，文档之间没有链接。它不需要join。

然而在一个社交网络中，没有什么能像这样独立存在。在任何时候，只要你看到什么东西看起来像是一个名字或者一幅图像，你就会希望能点击它，看看那个用户， 他们的个人资料，以及他们的帖子。TV show应用不是这样运作的。如果你位于巴比伦5(Babylon 5)的第一季第一集，你不会希望点击综合医院(General Hospital)的第一季第一集。

### 不要链接文档

我们开始在Diaspora代码中手工做这项烦人的MongoDB joins操作时，我们很清楚这只是麻烦的第一个迹象。这个迹象表明我们的数据实际是相互关联的，这个数据结构具有价值，而且我们正在违背文档数据库的基本概念。

不管你是否在复制关键数据（天啊），或者使用引用并在应用代码中执行join操作（天啊天啊），只要文档之间有链接，你就已经超越了MongoDB。当 MongoDB爱好者用不同的方式说“文档”的时候，他们的意思是那些你可以在一张纸上打印出来、拿在手里的东西。一个文档可能具有内部结构——标题 （headings）、子标题（subheadings）、段落（paragraphs）和页脚（footers）——但这并没有链接到其它文档。它是独 立的半结构化数据。

如果你的数据看起来是那样的，你只需文档。祝贺你！对Mongo而言它是一个好的用例。但是如果文档之间链接具有存在价值，那么实际上你拥有的就不是文 档。对你来说MongoDB就不是正确的解决方案。当然它也不是社交型数据的解决方案，在那种数据中文档之间的链接实际上是系统中最关键的数据了。

因此社交型数据不是面向文档的。那是否这就意味着它实际上是……关系型的呢？

## 又是那个词

当人们说“社交数据不是关系型数据”时，并不是他们说的意思。他们的意思是下面两个方面：

**1.“从概念上说，社交数据是一个比表集合更大的图谱。”**

这绝对是正确的。但是很少有概念自然的提及模式化表是标准化的。我们用结构化表示是因为它行之有效，这样做可以减少冗余，而且我们可以解决它变慢的问题。

**2.“当在非标准化的单文档结构中查询所有社交数据时是很快的。”**

这也是绝对正确的。当你的社交数据是按照关系型存储时，你为特定用户取出活动流将要在许多表中查询，并且当表越大速达越慢。然而，我们可以用简单的方式来解决这个问题。那就是缓存。

在牛津早些年的[数据库会议](http://allyourbaseconf.com/)中，我曾做过一个这样的报告，我强烈推荐你看看[Neha Narula](http://nehanaru.la/)谈论有关缓存的报告。无论如何，缓存让标准化数据存储变得复杂，但行之有效。我也曾看过缓存非标准化的活动流为一个文档结构，比如说MongoDB，它会让检索数据变得更快。但存在缓存失效的问题。

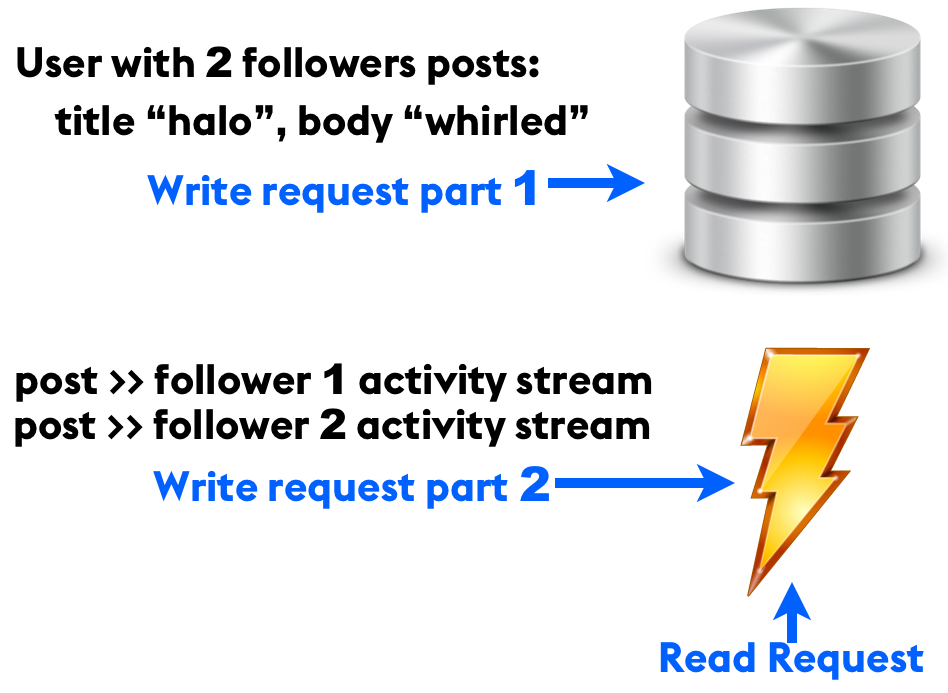
“在计算机学科中有两个头疼的问题：缓存失效和命名” Phil Karlton

他提出缓存失效的问题是很难解决的。Phil Karlton写过SSL版本3、X11和OpenGL，所以他对计算机了解的还是很多的。

## 缓存失效作为一个服务

那什么是缓存失效，为什么解决它很难？

众所周知，缓存失效就是缓存数据过期，它需要更新或是替换了。这里有个在网络应用中经常见的例子。我们有个后台存储器，典型的是PostgreSQL或MySQL，前台有一个缓存层，典型的结构是Memcached或Redis.请求读取用户的活动流时直接从缓存拿数据显然要比从数据库拿快的多。



典型的缓存和后台存储安装

应用的写操作更加复杂。让我们说说两个粉丝都写了一个消息的情况。首先发生的就是(第一部分)这些消息被复制和存储。一旦这些动作完成，后台将进入下一段工作(第二部分)将这些消息放入所有粉丝的活动流缓存中。

这种模式是很常见的。Twitter将近期活动用户的活动流都放入内存缓存中，当有粉丝发送消息时也将这些消息添加到缓存中。甚至是很小应用使用这样的活动流时也是这么做的(看看：7个表联合查询)。

回到我们的例子。当作者修改现存的帖子（post）时，更新过程在本质上与创建是一样的，唯一不同的是它不是增加到缓存，而是更新一个已经存在的条目。

如果步骤2的后台作业中途失败会怎样呢？机器重启了，网络线缆插头被拔掉了，应用重启了。在我们的工作中，不稳定是唯一不变的变量。当那些事情发生的时 候，你将会被缓存中的非法数据整崩溃。一些帖子的拷贝是旧的标题，而另一些拷贝却是新的标题。这是一个严重的问题，但是对于缓存而言，经常会有这种毁灭式 的情况。



经常的一种情况 >\_<

你完全可以从缓存中删除整个活动流记录，并从持久化的后台存储中重新生成它。这或许很慢，但至少这是可能的。

如果没有后台存储又会怎样呢？如果你跳过了步骤1呢？如果你仅仅只有缓存呢？

假如你只有MongoDB的话，它就是没有后台存储的一个缓存。它**将会**产生不一致。不是最终的一致——而一直都是纯粹的、彻头彻尾的不一致。就这一点而言，你没有选择。即使毁灭式的也没有。你**没有任何办法**重新生成一致状态的数据。

当Diaspora项目决定将关系型数据存储于MongoDB的时候，我们将数据库与缓存合并起来。数据库与缓存是非常不一样的两种事物。对于持久化、瞬态、复制、引用、数据完整性和速度，它们有完全不一样的思想。

### 转变

一旦我们理解了我们一不小心给数据库选择了一个缓存，那么我们是怎样使用这个缓存的呢？

好吧，这是一个价值百万美元的问题。但是我们已经回答了价值十亿美元的问题。在这篇文章中，我已经谈到了我们是如何使用MongoDB的，相对应的是，它 是如何设计其使用方法的。我已经谈过这一点了，就仿佛所有的信息都是显而易见的，只是Diaspora团队在做出选择之前没有做充足的研究。

但是这些东西一点也不显而易见。MongoDB文档告诉你它擅长什么，却没有强调它不擅长什么。这很好理解。所有项目都是这么做的。但是其结果是，这使我们花费了大约六个月，听到许多的用户埋怨，并且做了大量的调查，才由此断定我们使用MongoDB的方式不对。

没有什么别的办法，只有将数据从MongoDB中取出来，将它们迁移到一个关系型的存储设备，在此过程中要尽我们最大努力处理我们发现的不一致的数据。数据转变本身——由MongoDB导出，再导入到MySQL——非常简单明了。其中的技术细节，可以看看[《你所有的基础配置2013》中](https://speakerdeck.com/sarahmei/switching-data-stores-a-postmodern-comedy" \t "_blank)[的](https://speakerdeck.com/sarahmei/switching-data-stores-a-postmodern-comedy)幻灯片 。

### 损害

我们有八个月的生产数据，这大约对应于MySQL中的120万行。我们耗费了四个双周来开发这个转换代码，当我们开始实际实施的时候，主站有大约两个小时 的宕机时间。对于一个处于初期测试版的项目来说，这实在令人无法接受。我们应该缩短这个宕机时间的，但是却预估了八个小时的宕机时间，这样的话两个小时看 起来似乎还很漂亮。



还不坏

### 尾声

还记得电视剧（TV show）的应用吗？它是MongoDB的完美用例。每个剧集都是一个文档，完全独立的文档。它不引用任何东西，没有副本，而且数据没有不一致的可能。

距离开发约过了三个月后，电视剧应用仍然在MongoDB基础上很好的运行着。后来的一个星期一，在每周计划会议上，有委托人告诉我们，有个投资人想要一 项新的功能：当他们在某一集节目中看到某个演员的时候，他们想要可以点击该演员的名字，并看到这个人的整个电视职业生涯。他们想要该演员曾经出现过的所有 不同剧集的一个时间排序的列表。

我们将每个剧集保存为MongoDB中的一个文档，其中包含了所有嵌套的信息，包括 整个演员班底。如果同样的演员出现于两个不同的戏，甚至是出现于同一个剧集，他们的信息在两个地方都有保存。除了比较他们的名 字，我们没有办法识别出他们是否是同一个人。所以为了实现这个功能，我们必须搜索每个文档，找寻用户点击的演员，并删除重复记录。啊，对了。最起码，我们 需要删除一次重复记录，然后再维护演员信息的一个外部索引，就像任何其它的缓存一样，它同样也具有失效问题。

### ****你来看看这是怎么回事****

客户期待的功能是如此微不足道。如果数据已经在关系存储，它会一直在哪里。由于这是我们第一次尝试说服项目经理，客户并不需要它MongoDB。失败后， 我们提供了一些便宜的替代品，如链接到IMDB搜索演员的名字的产品。这个公司从广告赚钱，虽然如此，他们希望用户留在自己的网站上，而不是去上IMDB 。

此功能要求最终促使该项目的转换到PostgreSQL。当有更多的与客户交流后，我们意识到，客户企业看到把电视节目连接在一起很多价值。他们期望能够看到——正在看的节目的导演的其他节目。也希望能够看到——类似正在看的节目的其他本周发布的同一主题的节目。

这从根本上是一个沟通的问题，而不是技术问题。如果这些沟通已经提前发生了，如果我们花时间去真正了解客户端是怎么看到数据的和他们想要对数据做什么的话，我们可能会早些时候做这样一个沟通，那个时候有较少的数据，并且变更也较容易。

### ****一直在学习中****

我从经验学到：MongoDB的理想使用场景是比我们的电视数据更窄。唯一的事情是擅长的是存储任意个JSON数据。“任意”，在此背景下，意味着你不稀 罕什么是JSON里面。你甚至不看。没有模式，甚至没有一个隐含的模式，就犹如我们的电视节目数据。每个文件仅仅是一个blob数据，其内部数据是什么完 全不在意。

在RubyConf这个周末，我跑进康拉德欧文，谁提出这个用例。他用MongoDB的存储JSON的任意位的是来自客户通过一个API。这是合理的。这种帽子理论是完全不在意你的数据内容是否有意义。很有趣的是在应用程序中，你的数据很有意义的。

我已经听到很多人谈论到自己的web应用下探的MongoDB来替代MySQL或PostgreSQL。任何情况下，这都不是一个好主意。架构的灵活性听 起来像一个伟大的想法，但只有一次，它是真正有用的是当你的数据的结构没有任何价值。如果你有一个隐含的模式 - 这意味着，如果你期待返回JSON的数据 - 那么MongoDB是错误的选择。我建议采取看看PostgreSQL的hstore（现在比MongoDB的速度快的），并学习如何进行更改架构。他们 真的并不难，即使是在大表。

### ****寻找价值****

当你选择一个数据存储，应该了解最重要的事情就是你的数据在哪里，你的数据如何连接，你的数据的商业价值所在。如果你还不知道（这是正常的），那么选择不 会画你陷入了困境的数据存储。推JSON数据到你的数据库听起来很灵活，但真正的灵活性是很容易添加业务需求de 功能。

让有价值的东西做起来更加容易。

本文地址：<http://www.oschina.net/translate/why-you-should-never-use-mongodb>

原文地址：<http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/>