深入分析Redis特点及应用场景

六尺帐篷 (/u/f8e9b1c246f1)
2017.08.27 14:37* 字数 2703 阅读 116 评论 0 喜欢 7
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/15878374/notes/16324752)

> REmote DIctionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。
> Redis是一个开源的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。
> 它通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Map), 列表(list), 集合(sets) 和 有序集合(sorted sets)等类型。

Redis的特点：

Redis 与其他 key - value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。

Redis的优势：

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
- 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
- 原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。
- 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

Redis与其他key-value存储有什么不同?

- Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
- Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

Redis应用场景

1. 显示最新的项目列表

下面这个语句常用来显示最新项目，随着数据多了，查询毫无疑问会越来越慢。

```
SELECT * FROM foo WHERE ... ORDER BY time DESC LIMIT 10
```

在Web应用中，“列出最新的回复”之类的查询非常普遍，这通常会带来可扩展性问题。这令人沮丧，因为项目本来就是按这个顺序被创建的，但要输出这个顺序却不得不进行排序操作。

类似的问题就可以用Redis来解决。比如说，我们的一个Web应用想要列出用户贴出的最新20条评论。在最新的评论边上我们有一个“显示全部”的链接，点击后就可以获得更多的评论。

我们假设数据库中的每条评论都有一个唯一的递增的ID字段。我们可以使用分页来制作主页和评论页，使用Redis的模板：

- 每次新评论发表时，我们会将它的ID添加到一个Redis列表：

```
LPUSH latest.comments <ID>
```

- 我们将列表裁剪为指定长度，因此Redis只需要保存最新的5000条评论：

```
LTRIM latest.comments 0 5000
```

- 每次我们需要获取最新评论的项目范围时，我们调用一个函数来完成（使用伪代码）：

```
FUNCTION get_latest_comments(start,num_items): id_list = redis.lrange("latest.comments",star
```

这里我们做的很简单。在Redis中我们的最新ID使用了常驻缓存，这是一直更新的。但是我们做了限制不能超过5000个ID，因此我们的获取ID函数会一直询问Redis。只有在start/count参数超出了这个范围的时候，才需要去访问数据库。
我们的系统不会像传统方式那样“刷新”缓存，Redis实例中的信息永远是一致的。SQL数据库（或是硬盘上的其他类型数据库）只是在用户需要获取“很远”的数据时才会被触发，而主页或第一个评论页是不会麻烦到硬盘上的数据库了。

2. 删除与过滤

我们可以使用LREM来删除评论。如果删除操作非常少，另一个选择是直接跳过评论条目的入口，报告说该评论已经不存在。

有些时候你想要给不同的列表附加上不同的过滤器。如果过滤器的数量受到限制，你可以简单的为每个不同的过滤器使用不同的Redis列表。毕竟每个列表只有5000条项目，但Redis却能够使用非常少的内存来处理几百万条项目。

3. 排行榜相关

另一个很普遍的需求是各种数据库的数据并非存储在内存中，因此在按得分排序以及实时更新这些几乎每秒钟都需要更新的功能上数据库的性能不够理想。

典型的比如那些在线游戏的排行榜，比如一个Facebook的游戏，根据得分你通常想要：

- 列出前100名高分选手

- 列出某用户当前的全球排名

这些操作对于Redis来说小菜一碟，即使你有几百万个用户，每分钟都会有几百万个新的得分。

模式是这样的，每次获得新得分时，我们用这样的代码：

```
ZADD leaderboard
```

你可能用userID来取代username，这取决于你是怎么设计的。

得到前100名高分用户很简单：

```
ZREVRANGE leaderboard 0 99
```

用户的全球排名也相似，只需要：

```
ZRANK leaderboard
```

4. 按照用户投票和时间排序

排行榜的一种常见变体模式就像Reddit或Hacker News用的那样，新闻按照类似下面的公式根据得分来排序：score = points / time^alpha

因此用户的投票会相应的把新闻挖出来，但时间会按照一定的指数将新闻埋下去。下面是我们的模式，当然算法由你决定。

模式是这样的，开始时先观察那些可能是最新的项目，例如首页上的1000条新闻都是候选者，因此我们先忽视掉其他的，这实现起来很简单。

- 每次新的新闻贴上来后，我们将ID添加到列表中，使用LPUSH + LTRIM，确保只取出最新的1000条项目。
- 有一项后台任务获取这个列表，并且持续的计算这1000条新闻中每条新闻的最终得分。计算结果由ZADD命令按照新的顺序填充生成列表，老新闻则被清除。这里的关键思路是排序工作是由后台任务来完成的。

5. 过期项目处理

另一种常用的项目排序是按照时间排序。我们使用unix时间作为得分即可。

模式如下：

- 每次有新项目添加到我们的非Redis数据库时，我们把它加入到排序集合中。这时我们用的是时间属性，current_time和time_to_live。
- 另一项后台任务使用ZRANGE…SCORES查询排序集合，取出最新的10个项目。如果发现unix时间已经过期，则在数据库中删除条目。

6. 计数

Redis是一个很好的计数器，这要感谢INCRBY和其他相似命令。

我相信你曾许多次想要给数据库加上新的计数器，用来获取统计或显示新信息，但是最后却由于写入敏感而不得不放弃它们。

好了，现在使用Redis就不需要再担心了。有了原子递增（atomic increment），你可以放心的加上各种计数，用GETSET重置，或者是让它们过期。

例如这样操作：

```
INCR user: EXPIRE
user: 60
```

你可以计算出最近用户在页面间停顿不超过60秒的页面浏览量，当计数达到比如20时，就可以显示出某些条幅提示，或是其它你想显示的东西。

7. 特定时间内的特定项目

另一项对于其他数据库很难，但Redis做起来却轻而易举的事就是统计在某段特点时间里有多少特定用户访问了某个特定资源。比如我想要知道某些特定的注册用户或IP地址，他们到底有多少访问了某篇文章。

每次我获得一次新的页面浏览时我只需要这样做：

```
SADD page:day1:<page_id> <user_id>
```

当然你可能想用unix时间替换day1，比如time()-(time()%3600*24)等等。

想知道特定用户的数量吗？只需要使用

```
SCARD page:day1:<page_id>
```

需要测试某个特定用户是否访问了这个页面？

8. 实时分析正在发生的情况，用于数据统计与防止垃圾邮件等

我们只做了几个例子，但如果你研究Redis的命令集，并且组合一下，就能获得大量的实时分析方法，有效而且非常省力。使用Redis原语命令，更容易实施垃圾邮件过滤系统或其他实时跟踪系统。

9. Pub/Sub

Redis的Pub/Sub非常非常简单，运行稳定并且快速。支持模式匹配，能够实时订阅与取消频道。

10. 队列

你应该已经注意到像list push和list pop这样的Redis命令能够很方便的执行队列操作了，但能做的可不止这些：比如Redis还有list pop的变体命令，能够在列表为空时阻塞队列。

11. 缓存

Redis的缓存部分值得写一篇新文章，我这里只是简单的说一下。Redis能够替代memcached，让你的缓存从只能存储数据变得能够更新数据，因此你不再需要每次都重新生成数据了。

六尺帐篷 (/u/f8e9b1c246f1)
写了 241233 字，被 15271 人关注，获得了 1455 个喜欢
(/u/f8e9b1c246f1)