链滴

Redis 6 客户端缓存

作者：zeekling

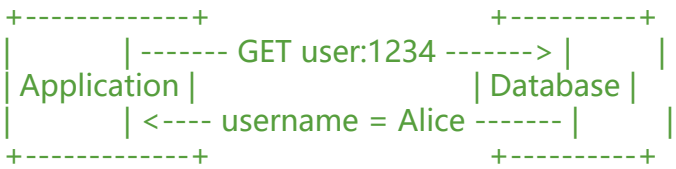
原文链接：https://ld246.com/article/1609680435219

来源网站：链滴

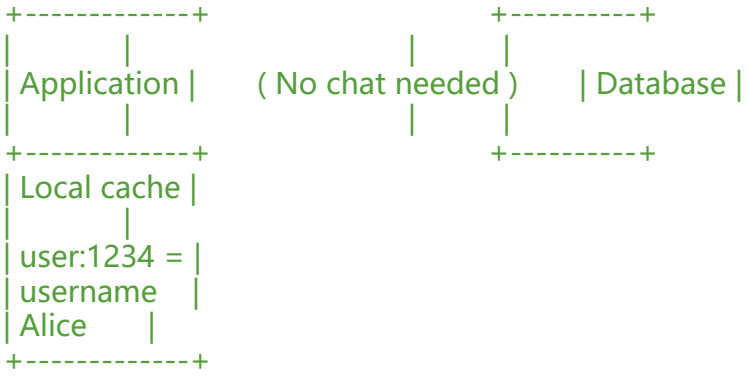
Redis服务器辅助的客户端缓存

客户端缓存是一种用于创建高性能服务的技术。它利用应用服务器中的可用内存，这些服务器通常是数据库节点不同的计算机，以便将数据库信息的某些子集直接存储在应用程序端。

通常当需要一些数据时，应用服务器会向数据库询问这些信息，如下图所示：



当使用客户端缓存时，应用程序将直接在应用程序内存中存储流行查询的答复，以便以后可以重用这答复，而无需再次联系数据库。



虽然用于本地缓存的应用程序内存可能不是很大，但是访问本地计算机内存所需的时间比请求数据库类的网络服务要小几个数量级。由于经常非常频繁地访问同一小部分数据，因此这种模式可以大大减应用程序获取数据的延迟，同时也减少数据库端的负载。

此外，有许多数据集中的项很少更改。例如，社交网络中的大多数用户帖子要么是不可变的，要么很被用户编辑。再加上一个事实，通常只有一小部分的帖子非常受欢迎，要么是因为一小部分用户有很关注者，要么是因为最近的帖子有更多的可见性，这就很清楚为什么这种模式非常有用。

通常，客户端缓存的两个主要优点是：

- 数据可用，延迟非常小。
- 数据库系统接收的查询较少，允许使用较少的节点为同一数据集提供服务。

存在问题

上述模式的一个问题是如何使应用程序所保存的信息无效，以避免向用户呈现过时的数据。例如，在面的应用程序本地缓存用户：1234信息，Alice可能会将她的用户名更新为Flora。然而，应用程序可继续为用户1234提供旧用户名。

根据我们所建模的应用程序的具体情况，这个问题并不是什么大问题，因此客户机只会使用固定的最“生存时间”来存储缓存的信息。一旦经过给定的时间，信息将不再被视为有效。在使用Redis时，复杂的模式利用Pub/Sub系统向监听的客户端发送无效消息。这是可以实现的，但是从所使用的带宽角度来看，这是非常棘手和昂贵的，因为这样的模式通常涉及到向应用程序中的每个客户端发送无效息，即使某些客户端可能没有无效数据的任何副本。此外，每个更改数据的应用程序查询都需要使用P

BLISH命令，这会使数据库花费更多的CPU时间来处理该命令。

不管使用什么模式，有一个简单的事实：许多非常大的应用程序实现某种形式的客户端缓存，因为这拥有快速存储或快速缓存服务器的下一个逻辑步骤。为此，redis6实现了对客户端缓存的直接支持，使该模式实现起来更简单、更易访问、更可靠、更高效。

客户端缓存的Redis实现

Redis客户端缓存支持称为跟踪，有两种模式：

- 在默认模式下，服务器会记住给定客户机访问的密钥，并在修改相同的密钥时发送无效消息。这会耗服务器端的内存，但只会为客户端可能在内存中拥有的一组密钥发送无效消息。
- 在广播模式下，服务器不会试图记住给定客户机访问的密钥，因此这种模式在服务器端根本不消耗何内存。相反，客户机订阅密钥前缀，如object:或user:，并且在每次碰到与该前缀匹配的密钥时都收到一条通知消息。

回顾一下，现在让我们暂时忘掉广播模式，集中讨论第一种模式。稍后我们将更详细地描述广播。

- 如果需要，客户端可以启用跟踪。连接在未启用跟踪的情况下启动。
- 启用跟踪后，服务器会记住每个客户端在连接生存期内请求的密钥（通过发送关于这些密钥的read令）。
- 当某个客户端修改了某个密钥，或者因为该密钥具有相关的过期时间而被逐出，或者由于maxmemry策略而被逐出时，所有启用了跟踪且可能缓存了该密钥的客户端都将收到一条无效消息。
- 当客户机接收到无效消息时，它们需要删除相应的密钥，以避免提供过时的数据。

这是协议的一个例子：

```
Client 1 -> Server: CLIENT TRACKING ON
Client 1 -> Server: GET foo
(The server remembers that Client 1 may have the key "foo" cached)
(Client 1 may remember the value of "foo" inside its local memory)
Client 2 -> Server: SET foo SomeOtherValue
Server -> Client 1: INVALIDATE "foo"
```

从表面上看，这看起来很不错，但是如果你认为，在每一个长时间连接的故事中，有10k个连接的客端都需要数百万个密钥，那么服务器最终会存储太多的信息。为此，Redis使用了两个关键思想来限服务器端的内存使用量，以及处理实现该功能的数据结构的CPU成本：

- 服务器会记住可能已在单个全局表中缓存给定键的客户端列表。这个表叫做失效表。这样的无效表以包含最大数量的条目，如果插入了一个新的键，服务器可以通过假装该键被修改（即使没有修改）并向客户端发送无效消息来逐出旧条目。这样做，它可以回收用于此密钥的内存，即使这将迫使拥有钥本地副本的客户端将其逐出。
- 在失效表中，我们实际上不需要存储指向客户端结构的指针，这将在客户端断开连接时强制执行垃回收过程：相反，我们所做的只是存储客户端ID（每个Redis客户端都有一个唯一的数字ID）。如果个客户机断开连接，信息将随着缓存槽的失效而逐渐被垃圾回收。
- 只有一个键名称空间，不除以数据库编号。因此，如果一个客户机正在缓存数据库2中的key foo，其他一些客户机更改了数据库3中key foo的值，那么仍然会发送一条无效消息。通过这种方式，我们以忽略数据库编号，从而降低内存使用量和实现复杂性。

双连接方式

使用Redis 6支持的新版Redis协议RESP3，可以在同一连接中运行数据查询和接收失效消息。然而，多客户端实现可能更喜欢使用两个独立的连接来实现客户端缓存：一个用于数据，另一个用于无效消。因此，当客户端启用跟踪时，它可以指定通过指定不同连接的“客户端ID”将无效消息重定向到另个连接。许多数据连接可以将无效消息重定向到同一个连接，这对于实现连接池的客户端很有用。双接模型是唯一支持RESP2的模型（它缺乏在同一连接中复用不同类型信息的能力）。

我们将展示一个例子，这一次在旧的RRESP2模式下使用实际的Redis协议，如何完成会话，包括以下骤：启用跟踪重定向到另一个连接，请求密钥，以及在该密钥被修改后获得无效消息。

首先，客户机打开第一个将用于失效的连接，请求连接ID，并通过Pub/Sub订阅用于在RESP2模式下取失效消息的特殊通道（记住RESP2是通常的Redis协议，而不是可以与Redis一起使用的更高级的协）6使用HELLO命令）：

```
(Connection 1 -- used for invalidations)
CLIENT ID
:4
SUBSCRIBE __redis__:invalidate
*3
$9
subscribe
$20
__redis__:invalidate
:1
```

现在我们可以从数据连接启用跟踪：

```
(Connection 2 -- data connection)
CLIENT TRACKING on REDIRECT 4
+OK

GET foo
$3
bar
```

客户机可能决定在本地内存中缓存“foo” => “bar”。

另一个客户端现在将修改“foo”键的值：

```
(Some other unrelated connection)
SET foo bar
+OK
```

因此，失效连接将接收一条消息，使指定的键失效。

```
(Connection 1 -- used for invalidations)
*3
$7
message
$20
__redis__:invalidate
*1
$3
foo
```

客户端将检查这种缓存槽中是否有缓存的密钥，并将逐出不再有效的信息。

请注意，发布/订阅消息的第三个元素不是一个键，而是一个只有一个元素的Redis数组。因为我们发了一个数组，如果有一组键要失效，我们可以在一条消息中完成。

要了解RESP2使用的客户端缓存和用于读取无效消息的Pub/Sub连接的一个非常重要的事情是，为了用旧的客户端实现，使用Pub/Sub完全是一个技巧，但实际上消息并不是真正发送到一个通道并由订它的所有客户端接收。只有我们在CLIENT命令的REDIRECT参数中指定的连接才会真正接收Pub/Sub息，这使得特性更具可伸缩性。

当改为使用RESP3时，无效消息将作为推送消息发送（在同一连接中，或者在使用重定向时在辅助连中发送）（有关详细信息，请阅读RESP3规范）。

什么跟踪记录

默认情况下，客户机不需要告诉服务器它们正在缓存哪些密钥。服务器会跟踪只读命令上下文中提到每个键，因为它可以被缓存。

这样做的明显优点是不需要客户机告诉服务器它在缓存什么。此外，在许多客户机实现中，这正是您希望的，因为一个好的解决方案可以是使用先进先出的方法缓存所有尚未缓存的对象：我们可能希望存固定数量的对象，我们检索到的每一个新数据都可以缓存它，丢弃最旧的缓存对象。更高级的实现能会放弃使用最少的对象或类似对象。

请注意，不管怎样，如果服务器上有写流量，缓存槽将在这段时间内失效。一般情况下，当服务器假我们得到的东西也要缓存时，我们正在权衡：

- 当客户机倾向于用一个欢迎新对象的策略缓存许多东西时，这种方法更有效。
- 服务器将被迫保留有关客户端密钥的更多数据。
- 客户端将收到关于它没有缓存的对象的无效消息。