第5章

键值数据库基础

5.1 从数组到键值数据库

键值数据库是最简单的一种 NoSQL 数据库。从名称可知，它根据键来存储数据，而这个键就是数据的标识符。

键值型数据结构可以看成是一种比较复杂的数组型数据结构。数组本来是一种简单的数据结构，但由于放宽了对该结构的诸多限制，并为其补充了与数据存储有关的特性，因此，数组这个概念覆盖的范围被放大了，它现在可以包括很多有用的数据结构，诸如关联数组、缓存以及键值数据库等。

5.1.1 数组

计算机专业的学生首先接触的数据结构可能就是数组。除了整数和字符串这样的变量外，数组应该是最简单的一种变量形式。数组是由整数值、字符或布尔值构成的有序列表，其中每个值都与一个整数下标（也称为索引）相关联，这些值的类型相同。图 5-1 所示为含有 10 个布尔元素的数组。

1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	True
9	False
10	True

图 5-1 含有 10 个布尔元素的数组

读取和设置数组元素值的具体代码与使用的编程语言有关。例如，采用以下写法来读取 exampleArray 的首个元素（其下标通常是 0 而不是 1）。

```
exampleArray[0]
```

要设置数组中某个元素的值，需要先写该元素的代码，然后写一个赋值符号（“=”号），然后写出给该元素所赋的新值。举例如下。

```
exampleArray[0] = 'Hello world.'
```

上面这行语句可以把 exampleArray 数组的首个元素设为字符串值“Hello world.”。数组中的其他元素也可以用以下语句来赋值。

```
exampleArray[1] = 'Goodbye World.'
exampleArray[2] = 'This is a test.'
exampleArray[5] = 'Key-value database.'
exampleArray[9] = 'Elements can be set in any order.'
```

exampleArray 数组中的每个元素都是由字符构成的字符串，不能把该数组的元素设为其他类型的值。由此可见，使用数组会有以下两项限制。

(1) 下标只能是整数。

(2) 所有的值必须是同一类型。

但是，有时人们可能需要使用一种不受这两项限制的数据结构。

5.1.2 关联数组

关联数组和普通数组一样，也是一种数据结构，但它的下标并不限于整数，而且也不要求所有的值都必须是同一类型。比如，可以用以下语句来操作关联数组。

```
exampleAssociativeArray['Pi'] = 3.1415
exampleAssociativeArray['CapitalFrance'] = 'Paris'
exampleAssociativeArray['ToDoList'] = {'Alice':'run
    reports; meeting with Bob':'Bob':'order inventory;
    meeting with Alice'}
exampleAssociativeArray[17234] = 36648
```

关联数组是对数组概念的泛化，它的标识符和元素值不像普通数组那样严格。它还有一些别名，如字典、映射、哈希(散列)映射(哈希图)、哈希表、符号表等。

从以上代码段可以看出，关联数组的键(相当于数组的下标或索引)既可以是字符串，也可以是整数。有些编程语言或数据库还允许使用数值列表等更复杂的数据结构来做键。

关联数组中值的类型也可以彼此不同。在以上例子中，数组里的元素值分别是实数、字符串、列表及整数等不同类型。与这些元素值相关联的标识符一般称为键。其实，关联数组就是键值数据库在底层所使用的基本数据结构。

5.1.3 缓存

数据存储在概念上和数据库类似，有时也用来指代包含数据库在内的各种数据保存形式。缓存也是一种键值数据存储机制。

键值数据库是根据关联数组这一概念构建的。许多键值型数据存储机制会把数据副本保存在硬盘或闪存盘等长期存储介质中，而另外一些键值型数据存储机制则会在内存中保留数据。程序可以通过后一种数据存储机制快速获取数据，这要比通过磁盘驱动器获取数据更迅捷。初次获取数据时，可以通过 SQL 查询语句从磁盘中的关系数据库里查出想要的结果，然后把查到的结果与相关的一组键名同时存储在缓存里，这些键名在缓存中是唯一的。

如果程序比较简单，每次只需记录一位顾客的信息，就可以使用字符串变量来保存该

顾客的姓名及地址。若程序需要同时记录多位顾客及其他一些实体，使用缓存会更合适。

内存中的缓存是一种关联数组。从关系数据库里获取一些值后，可以根据每个值来创建对应的键，并把这些键值对放到缓存中。对于每位顾客的每项数据来说，若保证这些键名彼此不重复，一种办法是把某个独特的标识符与该项数据的名称合起来当作键名。比如，以下语句可以把从数据库里获取的信息放入内存的缓存里面。

```
customerCache['1982737:firstname'] = firstName
customerCache['1982737:lastname'] = lastName
customerCache['1982737:shippingAddress'] = shippingAddress
customerCache['1982737:shippingCity'] = shippingCity
customerCache['1982737:shippingState'] = shippingState
customerCache['1982737:shippingZip'] = shippingZip
```

由于 customerID 的值(1982737)作为键名的一部分，因此这个缓存可以存放许多顾客的数据，不需要给每位顾客的每一组数据都单独起一个变量名，而是可以把它们全部放在 customerCache 这个关联数组之中。程序查询顾客的数据时，一般会先试着从缓存里获取，如果缓存中找不到，再去查询数据库。

5.1.4 键值数据库

键值数据库又称键值对数据库，是指能够持久保存数据的键值存储机制。对于需要多次查询数据库的应用程序来说，缓存能够提升效率。键值数据存储机制如果还能够把数据持久地保存在磁盘、闪存盘等其他长期存储设备之中，将会变得更为有用。这样的键值存储机制既具备高效的缓存，又带有能够持久存放数据的数据库。如果开发者想要更方便地存储并获取数据，而不追求表格或数据网络等较为复杂的数据结构，就可以考虑使用键值数据库。

制定好键名的规范后，可以编写一系列函数，以便在键值数据库中模拟表格的创建、读取、更新及删除操作。比如，以下这个用伪代码写成的 create 函数就可以模拟在表格中创建新行的操作。

```
define addCustomerRow(p_tableName, p_primaryKey,
    p_firstName, p_lastName, p_shippingAddress,
    p_shippingCity, p_shippingState, p_shippingZip)
        begin;
            set [p_tableName + p_primary + 'firstName'] = p_firstName;
            set [p_tableName + p_primary + 'lastName'] = p_lastName;
            set [p_tableName + p_primary + 'shippingAddress'] =
                p_shippingAddress;
            set [p_tableName + p_primary + 'shippingCity'] =
                p_shippingCity;
            set [p_tableName + p_primary + 'shippingState'] =
                p_shippingState;
            set [p_tableName + p_primary + 'shippingZip'] =
```

```
            p_shippimgZip;
    end;
```

完成读取、更新及删除操作的函数，写起来也很简单。

以键值数据库为基础，开发者很容易就能实现网络状或表格状的数据结构。可以制定一种键名规范，把表格名称、主键值以及属性名称合起来当作键名，以保存与之关联的属性。举例如下。

```
Customer:1982737:firstName
Customer:1982737:lastName
Customer:1982737:shippingAddress
Customer:1982737:ghippingCity
Customer:1982737:shippingState
Customer:1982737:shippingZip
```

5.2 键值数据库的重要特性

各种键值数据库都具有三个重要特性，即简洁、高速和易于缩放。为了实现这三个特性，数据库必须接受一些限制。

5.2.1 简洁：不需要复杂模型

键值数据库使用了功能极其简单的数据结构，因为常常用不到关系数据库提供的附加功能。比如，文字处理软件 Microsoft Word 具备很多强大的功能，它提供了一堆文本格式化选项，具备拼写检查及语法检查功能，还可以和引用资料管理器与参考书目管理器等工具相集成。如果要写一部书或一篇较长的论文，就应该使用这种功能丰富的文字处理软件。但是，如果只想在手机上编辑一份仅含 6 个条目的待办事务列表，那么一个简单的文本编辑器就足以完成。

一般来说，开发者都用不到表格的 join 操作，也不会同时查询数据库里的多种实体。要用数据库保存与购物车有关的数据，使用键值数据库会更简单一些。写好程序后，如果还需要在键值数据库中保存其他一些属性，可以直接把相关的代码添加到程序中。新的属性也可以直接添加到键值数据库中。

键值数据库使用的数据模型非常简单，操作数据的代码写起来也很容易。想操作某个键值对时，向键值数据库提供键名，数据库就会告之与该键相关联的值。

键值数据库使用起来比较灵活，规则也比较宽松。例如，以下这段代码同时使用了数字及字符串这两种值来做顾客标识符。

```
shoppingCart[cart:1298:customerID] = 1982737
shoppingCart[cart:3985:customerID] = 'Johnson, Louise'
```

5.2.2 高速：越快越好

由于使用了简单的关联数组做数据结构，又对提升操作速度进行了一些优化，因此，

键值数据库能够应对高吞吐量的数据密集型操作。

键值数据库既可以利用 RAM 实现快速的写入操作，又可以利用磁盘实现持久化的数据存储。程序如果修改了与某个键相关联的值，键值数据库就会更新 RAM 中的相应条目，然后向程序发送消息，告知该值已经更新。程序可以继续进行其他操作。程序执行其他操作时，键值数据库可以把最近更新的值写入磁盘之中。从应用程序更新该值起，至键值数据库将其存储到磁盘中为止，这中间只要不发生断电或其他故障，新值就可以顺利地保存到磁盘里。

由于数据库的大小可能会超过内存容量，所以键值数据库必须设法管理内存中的数据。对数据进行压缩，可以提升内存中所能保留的数据量。

当键值数据库得到一块内存之后，数据库系统有时需要先释放这块内存中的某些数据，以便存储新数据的副本。有很多算法都可以用来决定数据库所应释放的数据，其中最常用的一种叫做 LRU（最久未使用）算法。

5.2.3 缩放：应对访问量变化

键值数据库必须能在尽量不影响操作的情况下进行缩放，以应对 Web 应用程序和其他大规模应用程序的需求。可缩放性就是在服务器集群中根据系统的负载量随时添加或移除服务器的能力。对数据库系统进行缩放时，数据库对读取操作和写入操作的协调能力是一项很重要的性质。键值数据库可以采用不同的方式，针对读取操作和写入操作进行缩放。

5.3 键：有意义的标识符

键（key）是指向值的引用，它与地址的概念类似。键本身不是值，却提供了找寻与操作相关值的方式。键所具备的一项基本属性就是在所处的命名空间内，它的名称必须独一无二。

5.3.1 构造键名

在不同的键值数据库中，键可以表现为不同形式。Redis 等键值数据库可以用更复杂的结构做键，它支持的键类型包括字符串、列表、集、有序集、哈希映射、位数组。Redis 开发者把键值数据库称为数据结构服务器。

列表（list）是由字符串构成的有序集合。集（set）是由互不相同的元素以不特定的顺序构成的集合。有序集（sorted set）是由互不相同的元素按照特定顺序构成的集合。哈希（hash）映射是一种带有键值型特征的数据结构，能够把一个字符串映射为另外一个字符串。位数组（bit array）是由二进制整数构成的数组，可以使用与位数组有关的多种操作来分别处理其中的每一个二进制位。

构造键名时应该遵循一套固定的命名规范。比如，可以把实体类型、特定实体的独特标识符以及属性名称拼接起来，并用这个拼接好的字符串来充当键名。表示键名的字符串不应该太长，以避免使用很多内存，同时键名也不要起得太短，以避免引起歧义。

可以考虑把与属性有关的信息包含进来，使键名变得更有意义。构造键名时，可以把实体类型、实体标识符以及实体属性等信息拼接起来，举例如下。

```
Cust:12387:firstName
```

5.3.2 通过键定位相关的值

键值数据库的设计者更关心如何设计出实用的数据库，而不是简化访问数据所用的代码。可以用数字来定位相关的值，但这还不够灵活。应该能够使用整数、字符串乃至对象列表做键，办法就是用一种函数把整数、字符串或对象列表映射成独特的字符串或数字。像这样能够把某一类值映射为相关数字的函数，叫做哈希函数（也称为散列或杂凑）。

并非所有键值数据库都支持列表或其他复杂的结构。某些键值数据库对键的类型和长度作了比较严格的限制。

1. 哈希函数：将键名映射到相关位置

哈希函数是一种可以接受任意字符串，并能够产生一般不会相互重复的定长字符串的函数。例如，顾客购物车信息可以分别映射成表 5-1 中的各个哈希值。

表 5-1 键与哈希值之间的映射

键	哈 希 值
customer：1982737：firstName	e135e850b892348a4e516cfcb385eba3bfb6d209
customer：1982737：lastName	f584667c5938571996379f256b8c82d2f5e0f62f
customer：1982737：shippingAddress	d891f26dcdb3136ea76092b1a70bc324c424ae1e
customer：1982737：shippingCity	33522192da50ea66bfc05b74d1315778b6369ec5
customer：1982737：shippingState	239ba0b4c437368ef2b16ecf58c62b5e6409722f
customer：1982737：shippingZip	814f3b2281e49941e1e7a03b223da28a8e0762ff

虽然这些键的前缀都是“customer:1982737:”，但是映射出来的哈希值却有相当大的区别。哈希函数的一项特性就是要把键映射成看上去较为随机的一些值。在本例中，采用 SHA-1 哈希函数来生成这些哈希值。

表 5-1 中的哈希值都是十六进制的整数，其总长度均为 40 位，所以总共有大约 1.4615016×10^{48} 个不同的取值。对于使用键值数据库的应用程序来说，这已经足够用了。

2. 通过键避免重复写入

现在，来看如何用哈希函数所返回的数字把键映射到相关的位置上。为了简化整个讨论，我们只关注如何用函数返回的哈希码来确定与键相关的值应该存储在哪一台服务器上。在实现的键值数据库中，可能还要把键映射到服务器的磁盘或内存中更为具体的位置上。

假设有一个 8 台服务器的集群。哈希函数的好处就在于它返回的是个数字。由于写

入请求应该平均地分布在这 8 台服务器上，所以可以给每台服务器安排 1/8 的负载量。例如，可以把第一次写入请求交给 1 号服务器来处理，把第二次写入请求交给 2 号服务器来处理，把第三次写入请求交给 3 号服务器来处理，依此类推。不过，这种轮流处理的做法并没有发挥出哈希函数的优势。

要想利用函数返回的哈希值，一个办法是把该值与服务器的总数相除。有时，这个哈希值可以为服务器总数所整除。如果哈希函数返回的数字是 32，那么 32 除以 8 的余数就是 0。如果哈希函数返回 41，那么 41 除以 8 的余数就是 1。如果哈希函数返回 67，那么 67 除以 8 的余数就是 3。可以发现，除以 8 之后所剩的余数总是在 0～7 之间。于是，就可以把 0～7 这 8 个数字分别与这 8 台服务器联系起来。

5.4 值: 存放任意数据

键值数据库中的另一个组件就是值。键值数据库之所以简单，一部分原因在于它使用关联数组作为基本的结构，而这种结构本身就非常简单。NoSQL 数据库保存值的方式也很简单。

5.4.1 值无须明确类型

键值数据库的值没有固定的形态，它通常由一系列字节构成。值的类型可以是整数、浮点数、字符串、二进制大型对象(BLOB)，也可以是诸如 JSON 对象等半结构化的构件，还可以是图像、声音以及其他能够用一系列字节来表示的任意类型。

比如，可以把下面这个字符串与表示某顾客住址的键关联起来，代码如下。

```
'1232 NE River Ave, St. Louis, MO'
```

也可以不用字符串，而是采用列表的形式来存放住址信息，代码如下。

```
('1232 NE River Ave, St. Louis, MO')
```

另外，还可以用更有条理的 JSON 格式来存放此信息，代码如下。

```
{'Street: ':'1232 NE River Ave', 'City':'St. Louis',:
    'State':'MO'}
```

键值数据库对其中存储的数据结构基本不会做出限定。

大多数键值数据库都会对值的大小做出限定。例如，Redis 数据库可以支持长度为 512MB 的字符串值。以支持以 ACID 事务著称的 FoundationDB 数据库，其值的大小不能超过 100 000B。

不同的键值数据库也为值提供了不同的操作。所有的键值数据库至少都会支持对值进行读取及写入操作。有的键值数据库还支持其他一些操作，如把字符串追加到已有的某个值尾部，或是访问字符串中的任意一部分内容，这样可以提高效率。Radis 数据库支持另一种扩充功能，它可以根据值来编订全文索引，使开发者能够利用 API，通过查询语句来搜寻键和值。

在选择数据库的过程中，应该把应用程序的需求考虑进来，在多项特性之间做出权衡。某个键值数据库也许会支持ACID事务，但是却只允许使用较小的键与值。另一个键值数据库也许能够存放较大的值，但却规定只能用数字或字符串做键。

5.4.2 对值搜索时的限制

键值数据库对值的各种操作都是根据键来执行的。可以根据键获取相关的值，根据键设置相关的值，或是根据键删除相关的值。如果还要执行其他操作，如搜寻城市为“St. Louis”的地址信息，就需要开发者在应用程序里自己实现了。键值数据库并不支持用查询语言对值进行搜索。

5.5 键值数据库的数据建模

NoSQL数据库的标准化程度不高，不同的开发商及开源项目可能会在各自的NoSQL数据库中使用一些特有的专门词汇或数据结构。

5.5.1 数据模型和数据结构

数据库中的数据能够传达信息，而数据模型就是用来排列这些信息的一种抽象方式，它们与数据结构有所区别。

数据结构是一种有明确定义的数据存储结构，它一般要通过底层硬件中的某些元件来实现，这些元件通常是指随机存取存储器(RAM)或硬盘及闪存盘等持久化数据存储介质。例如，编程语言中的整数型变量可能会用4个连续的字节，也就是32个二进制位来实现。

含有100个整数的数组可以连续地存放在内存之中，数组里的每个元素都用4个字节来表示。数据结构都拥有一套处理本结构所用的操作。整数这种数据结构定义了加、减、乘、除等操作，而数组则提供了以下标为依据的读取操作和写入操作。

数据结构提供了一种宏观的排列方式，使开发者既不用去关注底层的内存地址，又无须通过这些地址在硬件层面上操作。

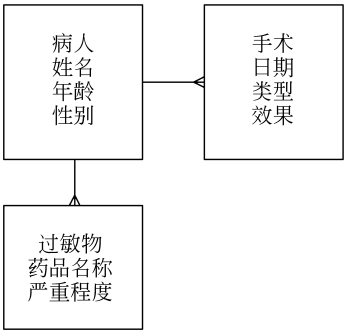

图5-2 数据模型是搭建在数据结构之上的一种抽象方式

数据模型也是用类似的方法抽象的，它是搭建在数据结构之上的一种排列和抽象方式(图5-2)。数据模型一般用来安排多种相关信息。对于管理客户信息所用的数据模型来说，可能会针对客户的姓名、住址、订单及支付记录进行建模。而对于临床数据库来说，则可能包含病人的姓名、年龄、性别、当前的处方、过去做过的手术、过敏情况以及其他一些与医疗有关的细节。实际上，记录这些数据更有效、更迅速的办法应该是使用数据模型与数据库。

数据模型中的元件会随着数据库类型的不同而有所不同。关系数据库是围绕表格来构建的。表格用来

存储与实体有关的信息，实体可以代表顾客、病人、订单或手术等事物。实体的属性用来记录特定实体的具体信息。属性可以是名称、年龄、配送地址等。在关系数据库中，表格是由很多列构成的，每一列都对应一项属性。表格内的每行则对应于实体的每个实例，如某位具体的顾客或病人。

设计数据库的时候，软件工程师会选择一些数据结构来实现数据模型中的表格及其他元件。这就减轻了应用程序开发者的工作量，使他们不用再处理那些细节问题。在关系数据模型的设计中，逻辑数据模型与物理数据模型是有区别的。实体和属性是逻辑数据模型使用的术语，二者分别对应于物理数据模型中的表格和列。

5.5.2 命名空间

命名空间是由键值对构成的集合，它可以想象成由互不重复的键值对所组成的一个集、一个集合或一个列表，或者一个存放键值对的桶(bucket)。一个命名空间本身就可以构成一个完整的键值数据库，它是由键值对构成的集合，而这些键值对中的键名都不会重复，而键值对的值是可以相互重复的。

如果多个程序都使用同一份键值数据库，命名空间就比较有用了。因为只要不打算在程序间共享数据，就无须担心键名构造方式会和其他程序的命名方式相冲突，因为命名空间会给值加上一个默认的前缀。顾客管理团队可以创建名为 custMgmt 的命名空间，而订单管理团队则可以创建名为 ordMgnt 的命名空间。这样他们就可以把各自用到的键和值都保存在自己的命名空间里。

5.5.3 分区与分区键

把数据分割成多个命名空间是一种非常有用的规划方式，与之类似，也可以把集群划分成多个小单元(称为分区)。集群中的每个分区都是一组服务器，或是运行在服务器上的一组键值数据库软件实例，而数据库中的数据子集则会分别交由这些分区来处理。所选的分区方案应该尽量把负载平均分配到集群中的每一台服务器。

请注意，同一台服务器中也可能出现多个分区。如果服务器上运行着虚拟机，而每台虚拟机都各自形成一个分区，就会出现这种情况。此外，键值数据库本身也可以在一台服务器上运行分区软件的多个实例。

分区键就是决定数据值应该保存到哪个分区所用的键。对于键值数据库来说，每个键都会用来决定与之相关的值存放在何处。例如，文档数据库只会把文档中的某一个属性当成分区键。

在某些情况下，仅依靠键本身可能无法实现负载均衡。此时应该使用哈希函数。哈希函数可以把输入的字符串映射成定长的字符串，对于不同的输入字符串来说，映射出的字符串通常也不会互相重复。可以将哈希函数理解为一种映射方式，它能够把一套分布不均匀的键名映射成另外一套分布较为均衡的键名。

5.5.4 无模式的模型

无模式用来形容数据库的逻辑模型。使用键值数据库的时候，不需要在添加数据之

前率先定义好所有的键，也不需要指定值的类型。例如，可以用以下这样的键来直接存储顾客的全名，代码如下。

```
cust:8983:fullName = 'Jane Anderson'
```

假设不想把顾客的全名都保存在一个值里，而把名字和姓氏分开保存，只需要修改保存相关键值的所用语句就可以了，代码如下。

```
cust:8983:firstName = 'Jane'
cust:8983:lastName = 'Anderson'
```

把顾客全名保存在一个字符串里的那些键值对，与将名字和姓氏分开保存的那些键值对是可以共存的，它们之间不会出现问题。

修改了姓名的存储方式后，开发者当然还需要修改程序的代码，使程序能够同时处理这两种表现形式，或者使程序能够把其中一种形式全都转换成另外一种形式。

5.6 键值数据库的架构

键值数据库的架构是指与服务器、网络组件及协调多台服务器之间工作的相关软件有关的一系列特征。键值数据库用自己的一套术语来描述数据模型、架构以及实现层面的组件。键、值、分区及分区键是与数据模型有关的重要概念，而集群、环以及复制是涉及架构的重要话题。

5.6.1 集群

集群(cluster)是一系列相互连接的计算机，它们彼此之间可以相互配合，以处理相关的操作。集群可以是松散耦合的，也可以是紧密耦合的。在松散耦合的集群中，各台服务器相对独立，它们只需要在集群内进行少量的通信就可以各自完成很多任务。而在紧密耦合的集群中，服务器之间会频繁地通信，以便完成一些需要紧密协作才可以实现的操作或计算。键值数据库集群一般是松散耦合的。

在松散耦合的集群中，每台服务器(也称为节点)可以把自己所要处理的数据范围分享给其他服务器，也可以定期给其他服务器发送消息，以判断那些服务器是否还在正常运作。这种互相传送消息的做法可以检测出发生故障的服务器节点。当某个节点出现故障时，其他节点可以把那个节点的工作量承揽过来，以便响应用户的请求。

某些集群会设立一个主节点。比如，Redis 数据库的主节点会负责执行读取操作及写入操作，也会负责把数据副本复制到各个从节点之中。从节点只受理读取请求。如果主节点发生故障，那么集群中的其他节点会选出新的主节点。若从节点发生故障，则集群中的其他节点仍然照常受理读取请求。还有一些集群是无主式的。例如，Riak 数据库的所有节点就都可以支持读取操作及写入操作。如果其中某个节点出现故障，其他节点会分担那个节点所需处理的读取和写入请求。