第 2 章 Chapter 2

# 影响 SQL Server 性能的因素

数据库的性能取决于各方面的综合因素，涉及硬件、操作系统及软件（如 SQL Server）、用户设计等几个大的方面。其中，硬件方面又包括内存、CPU 及磁盘等重要因素，并且这几个因素往往又是互相影响的。例如，当服务器的物理内存不足时，就会产生大量的磁盘 I/O，给磁盘带来压力；当内存不足时，一些占用 CPU 资源较多的对象（例如执行计划）可能就无法被正常缓存在内存当中，需要使用大量的 CPU 资源来处理这些对象的计算，从而给 CPU 带来更大的压力。

处在硬件之上的便是操作系统，SQL Server 只能运行在 Windows 环境下，而 Windows 操作系统不仅有各种不同的版本（包括 Windows Server 2000、Windows Server 2003、Windows Server 2008 及相关的子版本，诸如 SP1、SP2 等），还根据 CPU 不同而区分为 32 位和 64 位操作系统。这些版本的差异，会导致 SQL Server 只支持部分功能，同时性能优化的手段也有所差异。

SQL Server 也有诸多不同的版本，包括 SQL Server 2000、SQL Server 2005、SQL Server 2008、SQL Server 2008 R2、SQL Server 2012 及最新的 SQL Server 2014 等；从应用范围来分，又可以分为开发版、标准版本企业版数据中心版本等，这些不同的版本在功能和内部实现上都有所差异（详细的差异比较请参考 SQL Server 联机丛书），这些差异决定了 SQL Server 支持的最大内存、CPU 及数据文件的大小等，对此本章后续会做详细介绍。

影响数据库性能的因素，除了上面列出的因素外，还包括开发人员、DBA 等，比如表结构的设计、SQL 语句的编码等。据相关数据统计，75% 左右的性能问题由人为因素引起，因此，提升 SQL 编码质量是保证数据库高性能的重要因素。

## 2.1 服务器硬件

### 2.1.1 内存

在 SQL Server 的数据库引擎中，许多重要环节都需要使用到内存。SQL Server 日常维护中最值得关注内存的两个部分为执行计划缓存和数据缓存。

#### 1. 执行计划缓存（Plan Cache）

执行 T-SQL 语句时（例如，通过查询管理器（SQL Server Management Studio，SSMS）执行查询语句），待数据库引擎接收到需要执行的语句后，首先会经过一系列复杂的计算和分析，得到相应的执行计划，然后再根据执行计划进行各种操作。由于执行计划的计算和分析需要的 CPU 资源比较多，所以很容易引起服务器的 CPU 资源紧张。为了解决这个问题，SQL Server 将分析生成好的执行计划缓存起来，称之为执行计划缓存（Plan Cache），后面在介绍执行计划时，会对其进行更详细的介绍。

#### 2. 数据缓存（Data Cache）

相较于在内存中读 / 写数据的速度，在磁盘中读 / 写数据就实在太慢了。如果 SQL Server 在进行数据读 / 写时都是基于磁盘的，那么 SQL Server 的所有操作都可能慢到令人发指。为了避免出现这样的问题，SQL Server 的数据操作都是通过内存来实现的。

数据库在获得执行计划后，将会依据执行计划中所对应的操作步骤一步一步地来执行。若遇到数据访问操作符，则会首先检查对应的数据是否已经在数据缓存中，如果没有，才会通过数据读取模块从磁盘中读取数据，并将这些数据缓存到缓存中，方便之后的语句从缓存中获取数据。

如果数据库内存不足，数据库引擎会依据最近最少使用的算法（Least Recently Used，LRU）将缓存中的数据清理，然后将需要的数据从磁盘中读取并缓存到数据缓存中。这会引起大量的磁盘 I/O，并且导致执行语句的执行效率降低，若大批量的语句出现这样的情况，可能会导致服务器 CPU 占用率飙高。因此，在诊断服务器性能时，如果 CPU 长期处于繁忙状态，并且磁盘 I/O 偏高，不妨分析一下服务器的内存是否达到瓶颈了。

### 2.1.2 CPU

SQL Server 在进行任务调度、执行计划分析、排序等计算时都需要使用大量的 CPU 资源。一般情况下，CPU 占用资源在 30% 上下波动时，表示当前服务器较空闲，当达到 50% ~ 60% 时，表示当前服务器较繁忙，如果达到 80%，则表示服务器现在非常繁忙，此时就需要仔细地调查一下，是什么原因导致 CPU 的使用率过高，并且需要采取相应的手段处理。通常情况下，一些执行频率较高，并且性能不理想的语句会造成这样的问题；少数情况下是由于服务器资源达到一定的瓶颈而引起的。

### 2.1.3　磁盘 I/O

SQL Server 基于内存进行数据操作，以避免磁盘造成的数据读取缓慢，但这并不意味着数据库服务器对于磁盘的 I/O 能力就没有要求了。数据库引擎会定期地将缓存中的数据写入磁盘，在数据写入磁盘的过程中，为了确保数据的一致性，SQL Server 引入了轻量级的锁机制闩锁（即 Latch，在后续章节中会介绍到 Latch 的互斥）。在缓存数据写入磁盘过程中，缓存中的数据页将会被加上闩锁，以避免其他会话修改数据。当磁盘的读 / 写能力较弱时，同样会造成语句性能低下等问题。磁盘 I/O 除了会影响数据缓存的读 / 写外，还会影响数据库日志的记录。为了确保数据库的基本特性（ACID，在介绍数据库事务时会详细介绍），SQL Server 还引入了日志的机制，并且日志记录磁盘是几乎实时的，当事务提交时，事务日志便会写入磁盘，如果磁盘的 I/O 能力较低，同样会影响到语句的执行速度。

通常情况下，会保守地估计磁盘的一次 I/O 时间为 10 毫秒。如果远大于这个值，那磁盘的 I/O 能力就太差了，需要考虑更换磁盘来提高磁盘 I/O 的效率。

### 2.1.4　网络带宽

SQL Server 服务为服务进程，它与客户端程序为两个不同的进程。这两个进程间通过在 SNI（SQL Server Network Interface）协议层上建立网络连接进行通信，通信过程中使用“表格格式数据流”（TDS）格式的数据包进行数据传输。

SQL Server 服务器建立被称为“TDS 端点”（EndPoint）的对象。TDS 端点在安装 SQL Server 时由 SQL Server 安装，同时也可以通过 CREATE ENDPOINT 创建及使用 ALTER ENDPOINT 修改。

客户端通过 SQL Server Native Client 与服务器端进行通信，客户端与 SQL Server 服务端使用相同的网络协议（也就是说，在 SQL Server 服务端及 SQL Server Native Client 中都默认封装好了相同的协议），这些网络协议有以下四种：

- 共享内存。该协议默认是开启的。客户端与服务端在同一机器中时，可以使用该协议。通常指定服务器地址为 lpc：<servername>[\<instancename>]，机器名或机器名 \ 实例名会使用共享内存。
- TCP/IP。在 TCP/IP 协议开启时，可以使用该协议。客户端与服务端在不同机器或本机均可以使用该协议。通常指定服务器地址为 IP 地址时使用。
- 命名管道。启用 NamePipe 协议后，在客户端连接地址使用 np:\\computername\pipe\pipename 指定使用命名管道，本机使用 localhost、(local) 时也会使用命名管道。
- VIA。启用 VIA 协议后，在客户端指定 via:servername[\instancename],<nicnumber>:<port> 连接字符串会使用 VIA 协议。

在进行客户端与数据库引擎的指令发送和数据读取过程中，数据拷贝（Builk Copy 等）同时也包括一些高可用性（如镜像、Service Broker 等）过程，网络间的传输速率将直接影响到 SQL Server 的运行效率。

## 2.2 SQL Server 版本对性能的限制

SQL Server 不同的版本对内存、CPU 和数据文件的大小会有一些不同的限制，并且其收费标准也不同。因此，在选择数据库版本时，需要根据业务系统的特点及成本来选择最合适的版本。表 2-1 列出了几个主要版本的差异。

表 2-1 SQL Server 版本间的差异

| 版本 | 最大内存 | 最大 CPU | 最大数据库文件大小 |
|---|---|---|---|
| 企业版 | 操作系统支持的最大值 | 操作系统支持的最大值 | 524PB |
| 标准版 | 64GB | 限制为 4 个插槽或 16 核，取二者中的较小值 | 524PB |
| Web 版 | 64GB | 限制为 4 个插槽或 16 核，取二者中的较小值 | 524PB |
| Express | 1GB | 限制为 1 个插槽或 4 核，取二者中的较小值 | 10GB |

通常在选择新服务器前，需要清楚该服务器对应的业务逻辑及具体处理能力的要求，从而选择所需资源的数量，再选择对应的数据库版本。

## 2.3 SQL Server 系统的配置

### 2.3.1 内存配置

SQL Server 常用的内存配置选项有两个，即最大服务器内存（max server memory）和最小服务器内存（min server memory）。

#### 1. 最大服务器内存

该选项表示 SQL Server 的 Buffer Pool 最大可使用的内存量，以 MB 为单位，默认值为 2147483647MB。

当该选项被配置为 0 或超过当前系统最大内存值时，将使用系统最大内存量；当设置为小于当前系统的最大内存值，并且大于最小内存值时，SQL Server 实例在达到指定的最大内存量后，将不会继续扩大内存的使用量，当服务器上除了 SQL Server 服务外还部署了 IIS 服务时，为了避免 IIS 服务因为无法获得充足的内存资源而导致 Web 站点不稳定，可进行资源隔离，在这种情况下，设置 SQL Server 占用的最大服务器内存非常有效。

SQL Server 服务进程与操作系统间也存在内存资源分配的情况。当服务器内存不足以分配给操作系统用于远程登录或启动某些服务器进程时，有可能会使服务器进入假死状态。因此，在安装好一个新的数据库服务器后，建议充分评估数据库需要的最大内存使用量，为操作系统或其他服务程序预留足够的内存空间。

#### 2. 最小服务器内存

该选项表示 SQL Server 的 Buffer Pool 使用的内存量不得小于这个数值，同样以 MB 为单位，默认值为 0。

该配置常用来为 SQL Server 实例预留能够使用的内存。当服务器内存出现较大压力时，数据库会根据 SQL Server 的内存需求与服务器内存的压力情况，收缩数据库持有的内存量；当收缩内存达到当前配置值时，将不再收缩。该选项还可以与“锁定内存页”一起使用，从而将 SQL Server 数据缓存长期锁定在内存。这个选项比较“危险”，所以请在翻阅更多资料，并有深入了解以后再做配置。

下面是对数据库内存选项进行配置的两种方式：

- 通过 SQL Server 的系统存储过程 sys.sp_configure 进行配置。
- 通过第 1 章中图 1-4 所示的选项页进行配置。

### 2.3.2　CPU 配置

下面介绍 SQL Server 数据库与 CPU 相关的几个配置选项。

（1）affinity mask

该选项用于与 CPU 关联（只能应用于 0 ~ 31 编号的 CPU，即只能控制前 32 个 CPU，CPU 术数大于 32 后，需要使用 affinity64 mask 选项），可通过配置 CPU 关联掩码来配置使用哪一个 CPU。

示例：在某服务器中，有 8 个 CPU，当该选项配置为 3 时，其二进制掩码为 00000011，则表示 0 位和 1 位上对应的 CPU 将投入使用。

（2）affinity64 mask

与 affinity mask 的功能类似，只是该配置表示绑定第 33 ~ 64 个 CPU 的关联掩码。

（3）affinity I/O mask

该选项表示将 SQL Server 的磁盘 I/O 与指定的 CPU 子集绑定。请注意：该选项与 affinity mask 选项是互斥的，不可以把 anfinity mask 与 anffinity I/O mask 配置成使用相同的 CPU。

（4）affinity64 I/O mask

与 affinity I/O mask 类似，它表示的是绑定第 33 ~ 64 个 CPU 的关联掩码。

通常情况下，上面这 4 个选项并不需要做特别修改。同时，这几个选项对于 SQL Server 服务器的整体影响非常大，修改这几个选项需要十分谨慎。如果需要对默认配置做修改，请预先进行充分测试，以避免生产环境中出现不可预估的问题。

### 2.3.3　I/O 及数据文件配置

在数据库服务器初期规划的时候，应该根据业务系统数据的分布情况，考虑选择合适的物理存储方案，例如 RAID、SAN 等，并进行相应的压力测试。

针对高并发的数据库系统，建议使用如下配置方案：

- 将主数据库的数据文件拆分成多个文件。
- 将数据文件与日志文件存放在不同的物理磁盘，从而提高 I/O 的并发。

- 系统数据库文件，特别是 Tempdb 的数据文件要放在独立的物理磁盘，并将数据文件拆分成多个，建议与逻辑 CPU 的个数相同，以提高并发。

## 2.4 数据库结构的设计

### 2.4.1 好的性能出自好的设计

性能优化贯穿在整个软件生命周期，故而在进行需求分析和设计的时候就应该考虑数据库的性能。

举个例子：假设某电子商务网站需要记录订单的信息，每个订单可能会有若干个商品，同时订单有不同的状态，状态的处理包括了若干个流程。若此时新增一个表，命名为 Order_Bad，并将订单及订单中的商品全部都存放在 Order_Bad 这个表中，那么，当订单数量达到百万级时，订单中的商品可能就会达到千万级了。在这种情况下，若需要查找某些订单的状态，就不得不从这些大量的订单商品数据中去“吃力”地查找，并且由于表结构比较混乱，在为表创建索引的时候，将会有许多不确定因素，这会使得表的维护成本很高。若在进行表设计时使用的是两张主子级关系的表，那么在查找订单信息时，就只需要从这百万级量的数据中去检索数据，从而避免了从千万级表中去检索数据，性能自然而然地也就提升了。所以请牢记“好的性能出自好的设计”，且“优化贯穿着整个软件的生命周期”！

---

注意 笔者的日常维护工作有很大一部分就是审阅开发人员提交的表结构变更。下面分享一下相应的经验。

1）尽可能地添加数据完整约束，例如非空约束、默认值约束、check 约束、唯一约束、外键约束等，这些约束的添加将有助于数据库关系引擎分析执行计划。

2）使用尽可能小的字段类型，特别是大表，尽量小的空间将可以带来更佳的性能。

3）表结构的设计应考虑业务需求带来的操作（查询、更新、删除等）及频率，尽可能地使业务逻辑实现简洁，使用简单的 SQL 语句，可避免过多的表关联。

---

### 2.4.2 约束对性能的影响

SQL Server 提供的约束类型分别为默认值约束、Check 约束、唯一约束和外键约束。接下来详细介绍它们对性能的影响。

#### 1. 默认值约束

在新增数据时，若没有指定字段的值，则使用默认值约束中的定义自动填充数值。默认值约束只在新增数据时才会被调用，且它只会影响新增的行，对数据库引擎来说，它的影响几乎可以忽略不计。

2. Check 约束

在新增或修改数据时，会先判断新值是否符合 Check 约束的定义，若不符合则返回相应的警告信息。Check 约束可以指定一些较为复杂的表达式，在指定 Check 约束时，应使逻辑尽量简单。Check 约束的引入，相当于告诉查询优化器：“这个字段的值只有某些枚举值”。例如，“性别”字段，其约束只允许字段值有“Male”和“Female”两种值，当查询语句通过这个字段查找“Male”或“Female”时，就直接返回空数据，而不再对数据进行大量的检索操作，因为约束告诉查询优化器，过滤的这两个值“Male”与“Female”是不存在的。

3. 唯一约束

创建一个唯一约束会默认创建一个唯一索引，它在校验的过程中，是利用索引的树结构来检索的，在数据更新的时候会有一定的开销，不过这个开销非常小，也可以忽略。同时，唯一索引是筛选率非常高的一种索引结构，对于查询优化有极大的帮助。通常在一个表中，无论是无意义的自增值，还是有意义的业务字段，至少应该有一个唯一约束存在。

4. 外键约束

外键约束为了保证主、子表数据的完整性，在子表进行增、删、改等操作的时候，会同时校验主表的数据是否完整。这个过程中，会对主表加锁进行查询，这时主表的关联字段一定是主键或唯一键，它的校验速度非常快，不会有太大的问题。当子表进行删除操作的时候，外键约束会去校验子表的关联字段是否存在。此时，如果子表的关联字段没有索引，将会扫描整个子表，那会使开销剧增（如果子表非常大）。所以在添加外键关联的时候，需要同时检查子表中的关联字段是否有索引，如果没有，请手工创建它（有部分网友误以为 SQL Server 会自动创建索引，其实是不会的）。在某些情况下，比如子表是日志表或历史表，对于数据完整性并没有要求，此时请不要在上面加外键约束，因为这只会给系统带来额外的消耗。

### 2.4.3　适当的冗余

字段冗余是避免过多表关联的常用手段。

但是冗余带来的一个问题就是维护上的成本提升，所以在添加冗余字段前，不妨多做些权衡，想清楚是避免关联查询带来的好处多，还是维护成本高带来的坏处多。例如：某个查询是某网页的默认加载查询，这个查询每天的访问率是 10 万次，而字段的更新只是每天甚至是几天才会修改一次。这时，完全可以考虑使用冗余字段来提高查询的效率。

## 2.5　T-SQL 语句的编写

### 2.5.1　编写 T-SQL 语句的注意事项及小窍门

无论是 DBA 还是开发人员，在使用数据库时，不可避免地需要编写 SQL 语句。这里

将针对语句的编写分享一些注意事项及小窍门，在后续的章节中，将会涉及这些注意事项和小窍门的原理。

- 编写语句前，先明确已经完全理解了业务需求，并知道表的用途及用法。
- 确定业务需要用到的过滤字段能否使用索引，是否有必要在字段上添加索引。
- 不要对有索引的字段使用任何计算，包括函数。因为这会导致无法使用索引进行数据检索，从而导致扫描（表或索引）操作。
- 小表操作优先，以小表驱动大表，使其尽量使用 NESTED LOOP（NESTED LOOP 是表关联操作的一个物理操作方式，它使用 foreach 的方式以较小数据量的数据集为驱动，内嵌 foreach 循环较大的表进行对比，其效率比其他几个的关联操作高。这将在后续的章节中进行详细介绍）。
- 只查询需要的字段，避免使用“*”返回所有的字段。
- 尽量使用简单 SQL 语句来实现业务功能，如果功能过于复杂，可以考虑将其拆分成若干个简单 SQL 语句。

### 2.5.2 使用简单 SQL 语句

简单 SQL 语句，它可以定义为：

- 只简单地存在 2 ~ 4 个表的关联，或者只是单表查询。
- 没有复杂的过滤条件，只有 2 ~ 3 个条件判断，并且有一个过滤条件明确（即使用等号），可以使用索引查找操作。

使用简单语句，尽量不要使用过于复杂的语句，可以避免数据库查询优化器在分析执行计划时要获取过于复杂的信息，造成查询优化器选择不是最优的执行计划。同时，简单语句也方便程序员查看相应的业务逻辑，更方便后续的维护人员理解业务逻辑和代码。

根据笔者的经验，越是复杂的语句，在业务量较大的系统中，语句执行计划产生“变异”的几率会越高。“变异”的执行计划通常会造成大量的 I/O、CPU 及内存资源的压力，执行频率高的语句，甚至有可能导致数据库假死或宕机。

## 2.6 小结

本章是一个概括章节，简要介绍了影响数据库服务器性能的几个主要因素。包括硬件、软件，以及编码、设计上的一些注意事项。在绝大多数情况下，数据库性能不佳都是由于设计及编码引起的。在后续的章节中，将对本章提到的要点，进行更为详细地剖析，结合内部机制与示例进行介绍。