中午午休时，正好收到公司的培训邮件，由公司性能测试组的一名年轻的同事为我们带来压测相关的分享，这部分对俺这个以应用开发为主的程序员来说，感觉帮助很大。课上内容非常的通熟易懂，涉及了一般**应用接口**开发中主要的性能问题（不属于分布式大并发），实用性非常的强，本文将选取个人认为其中相对常见部分进行介绍，不足之处望大家指出，再次感谢那名牛X同事，嘿嘿。

压力测试(Stress Test)指模拟实际应用中的软硬件环境和相应系统负载情况，在此条件下，对被测系统进行长时间或超大负荷的运行，来测试系统的性能、可靠性、稳定性，因此也被称为负载测试。常见的压测工具有LoadRunner、JMeter等，前者是付费软件，内容比较复杂，适合专业的压测人员使用；后者简单免费，大部分的业务场景都够用了，非常适合开发人员自己进行压测，相关使用介绍请见：<http://www.cnblogs.com/wanliwang01/p/JMeter_Base.html>。

压力测试的相关指标非常的多，初学时很容易迷失在其中，接下来，将通过一个表格介绍**最常见的几个指标**。

|  |  |
| --- | --- |
| 指标 | 诠释 |
| 每秒事务数TPS(transaction per second) | 最关键的指标，每秒能承受的并发数，需要注意的是，这儿强调的是并发(比如一秒可以顺序处理10个事务，并不能称之为是并发数，需要注意)，这是最关键的生产指标 |
| 响应时间RT(Response Time) | 通常关注平均响应时间、和不同分位的响应时间，比如90%要多久，99%要多久 |
| 并发数/线程数 | 在不同的测试工具中，对于一次用户请求有不同的名称，认为是并发的请求数即可 |
| 检查点/断言 | 是对结果的检查，简单来说就是response的结果是否满意的问题 |
| 事务成功率 | 这个也是系统的关键指标，比如当有多事务失败时，系统即使没有宕机，也被认为压力存在问题 |

压测还需要注意相关因素的考虑，包括并发量的大小、测试场景的选择（单一场景或复合场景）、压测服务器的环境（是单机还是集群）、测试时间的选取（10-15的通过性测试或者是8小时以上的稳定性测试）、压测结果的分析等。根据不同的要求，压测可以分为基准测试、负载测试和稳定性测试等。

在整个压测过程中，除了需要关注压测工具的反馈外，还需要注意以下关注点：数据量(是百万级、千万级或更大)、CPU/内存（可以通过zabbix查看iis wp的情况、用jmx查看java应用的情况）、IO/网络速度、数据库(可以通过数据库的慢日志来查找问题)的情况等。

接下来进入本文的重点，即性能案例的分享，虽然都比较简单(简化了场景)，但在工作中常常会因为疏忽而遗漏，而造成比较大的影响，希望大家都能避免接下来的问题，每天都准时回家陪老婆孩子，哈哈。

**1.dotNet内存托管（内存泄漏）**

在应用开发中，我们常常会依赖于第三方组件（无论是本公司还是其他公司提供），部分的组件存在不完善的问题。比如一些托管资源并不会隐式回收，这时就需要手动的释放，比如Client.Close()。处理这类问题可以通过观察应用的内存使用，如果一段时间内服务没有很高负载，但内存消耗仍然高居不下时，往往是这类问题，可以选用单一场景，查看堆内存使用情况的方式来进一步定位。

**2.设计不合理**

通常与社交相关的场景，都涉及很大的数据量，这时如果产品设计不合理，就会出现资料大量消耗的问题，这类问题主要通过评审会议来发现。比如我要关注一个朋友，如果实时的将其所有的文字和图片信息都通过过来，就会有巨大的信息量，通过分页（部分）查询和异步同步的方式可以解决此类问题。

**3.JVM参数设置不合理**

这个主要和JVM的GC有关，如果没有设置合理的老生代和永久代的大小，就很容易触发Full GC(Global GC)，可以通过配置jvm相关参数来解决，在上线前一定要注意检查。

**4.数据库的隐式转化**

这个问题，对于.NET程序员来说，一点也不陌生，SQL Server非常的智能，能帮助我们优化SQL语句而避免全表扫描，但也因为其带来一些问题，比如字符串类型的隐式转换。当数据库的字段类型为char(20)时，如果我们将DAO层的DBType设置为String就会出现字符串类型的隐式转换，因为这儿会将nchar转化为char,这个操作会消耗数据库大量的性能，可以通过执行计划发现。因此，需要习惯将char对应到DBType.AnsiString，varchar对应到DBType.String.

**5. 特殊场景**

这儿的特殊指一般不容易发生，很难重现的场景，往往会出现在与配置相关的场景中。比如写10条配置信息到Redis，如果出现10个并发的情况，如果代码不完善，就可能在Redis中产生100条记录，这会明显影响系统的性能。由于这种情况，往往只会在初次配置时发生，因此很难排查，需要在日常代码的编写中，养成考虑并发问题的习惯。

**6.IIS Threads过多**

这部分我的印象比较深，刚开始学习多线程编程时，觉得非常的炫酷，因此偏向于起一个线程去处理耗时的操作，比如数据库相关操作。当系统调用频繁时即压力很大时，会创建非常多的新线程和数据库连接，最终导致iis中大量线程处于wait状态，即使请求数下降，线程数和系统消耗不能回落，这部分可以考虑使用单例模式解决，减少资源的消耗。

7.线程block

这部分需要提高代码能力，无论是使用系统管理的线程池或者是.NET中提供的异步编程模型，都可以得到一定的解决。一定要记住的是，即使是.NET线程，也是需要消耗很多系统资源的，在使用时一定要注意对其进行管理。记得的一个例子是，通过多线程写日志，当TPS从200变为300时，RT直接从30ms变为800ms，出现了数量级的变化，最后发现是因为写日志造成的block。在压测过程中，尤其要注意RT数量级的变化，如果出现，必须引起重视。

8.Java/dotNET反射

反射通常与框架有关，有时个人为了简化代码，也会自己编写一个小框架，这是一定需要性能问题，如果接口有一定的性能要求，且自身不能很好的使用反射时（主要熟悉反射元数据的缓存甚至动态发射元数据），还是推荐出点体力。

**9.mysql索引失效**

这个问题也非常的常见，由于SQLServer的SQL优化的强大，造成个人在编写SQL语句时常常不注意细节。但当使用MySQL时，就需要严格的按照SQL标准来编写查询代码了，不然就会出现索引失效的情况，比如组合索引不按照顺序来编写（遵循最左前缀匹配原则）、勿用函数（比如DateDiff等时间日期函数，可以通过应用程序计算的方式处理）等，这部分可以说是最常见的性能调优点了。

10.锁问题（间隙锁）

比如在一个事务中同时使用delete/update和insert语句，当出现并发状况时，会出现大量事务失败的情况，解决方案就是分析事务，尽可能将其分解到两个事务中。

11.Linux内核配置问题（与运维相关）

这部分与操作系统内核的配置有关，比如Linux默认的内核配置tcp连接是不能重用的，然而当并发量变大时，比如每秒4000TPS，就会出现大量连接Time\_wait的情况，如果继续积聚，就会消耗完所有的连接，最终造成服务不可用的情况。解决个问题只需要在网络中添加tcp\_tx\_used的配置即可，这是连接数就可以稳定在4000+左右，这儿想说的是，如果所有可能情况都排查了，就可以考虑操作系统级别的问题了哈。

路漫漫其修远兮，吾将上下而求索! ☺