**数据库连接池学习笔记（一）：原理介绍+常用连接池介绍**

**什么是连接池**

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个。

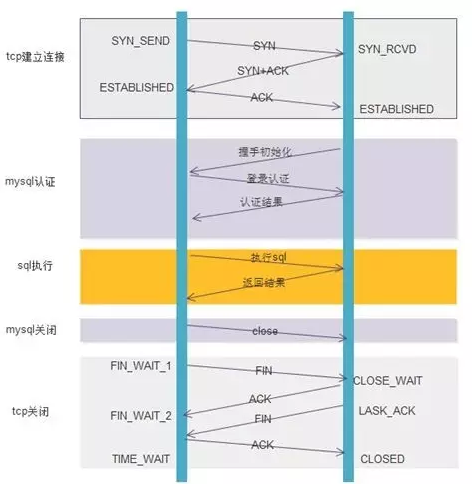
**为什么要使用连接池**

 数据库连接是一种关键的有限的昂贵的资源，这一点在多用户的网页应用程序中体现得尤为突出。  一个数据库连接对象均对应一个物理数据库连接，每次操作都打开一个物理连接，使用完都关闭连接，这样造成系统的 性能低下。 数据库连接池的解决方案是在应用程序启动时建立足够的数据库连接，并讲这些连接组成一个连接池(简单说：在一个“池”里放了好多半成品的数据库联接对象)，由应用程序动态地对池中的连接进行申请、使用和释放。对于多于连接池中连接数的并发请求，应该在请求队列中排队等待。并且应用程序可以根据池中连接的使用率，动态增加或减少池中的连接数。 连接池技术尽可能多地重用了消耗内存地资源，大大节省了内存，提高了服务器地服务效率，能够支持更多的客户服务。通过使用连接池，将大大提高程序运行效率，同时，我们可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

**传统的连接机制与数据库连接池的运行机制区别**

**不使用连接池流程**

下面以访问MySQL为例，执行一个SQL命令，如果不使用连接池，需要经过哪些流程。



不使用数据库连接池的步骤：

1. TCP建立连接的三次握手
2. MySQL认证的三次握手
3. 真正的SQL执行
4. MySQL的关闭
5. TCP的四次握手关闭

可以看到，为了执行一条SQL，却多了非常多我们不关心的网络交互。

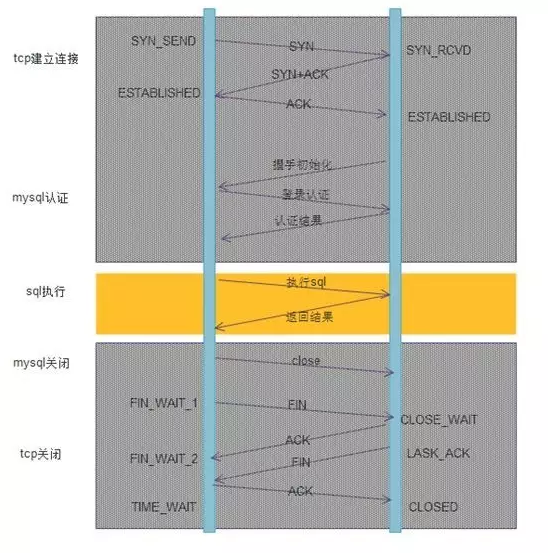
**优点：**

1. 实现简单

**缺点：**

1. 网络IO较多
2. 数据库的负载较高
3. 响应时间较长及QPS较低
4. 应用频繁的创建连接和关闭连接，导致临时对象较多，GC频繁
5. 在关闭连接后，会出现大量TIME\_WAIT 的TCP状态（在2个MSL之后关闭）

**使用连接池流程**



使用数据库连接池的步骤：

第一次访问的时候，需要建立连接。 但是之后的访问，均会复用之前创建的连接，直接执行SQL语句。

**优点：**

1. 较少了网络开销
2. 系统的性能会有一个实质的提升
3. 没了麻烦的TIME\_WAIT状态

**数据库连接池的工作原理**

连接池的工作原理主要由三部分组成，分别为

1. 连接池的建立
2. 连接池中连接的使用管理
3. 连接池的关闭

        第一、连接池的建立。一般在系统初始化时，连接池会根据系统配置建立，并在池中创建了几个连接对象，以便使用时能从连接池中获取。连接池中的连接不能随意创建和关闭，这样避免了连接随意建立和关闭造成的系统开销。Java中提供了很多容器类可以方便的构建连接池，例如Vector、Stack等。

        第二、连接池的管理。连接池管理策略是连接池机制的核心，连接池内连接的分配和释放对系统的性能有很大的影响。其管理策略是：

        当客户请求数据库连接时，首先查看连接池中是否有空闲连接，如果存在空闲连接，则将连接分配给客户使用；如果没有空闲连接，则查看当前所开的连接数是否已经达到最大连接数，如果没达到就重新创建一个连接给请求的客户；如果达到就按设定的最大等待时间进行等待，如果超出最大等待时间，则抛出异常给客户。

        当客户释放数据库连接时，先判断该连接的引用次数是否超过了规定值，如果超过就从连接池中删除该连接，否则保留为其他客户服务。

        该策略保证了数据库连接的有效复用，避免频繁的建立、释放连接所带来的系统资源开销。

        第三、连接池的关闭。当应用程序退出时，关闭连接池中所有的连接，释放连接池相关的资源，该过程正好与创建相反。

**连接池主要参数**

**使用连接池时，要配置一下参数**

1. 最小连接数：是连接池一直保持的数据库连接,所以如果应用程序对数据库连接的使用量不大,将会有大量的数据库连接资源被浪费.
2. 最大连接数：是连接池能申请的最大连接数,如果数据库连接请求超过次数,后面的数据库连接请求将被加入到等待队列中,这会影响以后的数据库操作
3. 最大空闲时间
4. 获取连接超时时间
5. 超时重试连接次数

**连接池需要注意的点**

**1、并发问题**

　　为了使连接管理服务具有最大的通用性，必须考虑多线程环境，即并发问题。这个问题相对比较好解决，因为各个语言自身提供了对并发管理的支持像java,c#等等，使用synchronized(java)lock(C#)关键字即可确保线程是同步的。使用方法可以参考，相关文献。

**2、事务处理**

　　我们知道，事务具有原子性，此时要求对数据库的操作符合“ALL-OR-NOTHING”原则,即对于一组SQL语句要么全做，要么全不做。   
　　我们知道当2个线程共用一个连接Connection对象，而且各自都有自己的事务要处理时候，对于连接池是一个很头疼的问题，因为即使Connection类提供了相应的事务支持，可是我们仍然不能确定那个数据库操作是对应那个事务的，这是由于我们有２个线程都在进行事务操作而引起的。为此我们可以使用每一个事务独占一个连接来实现，虽然这种方法有点浪费连接池资源但是可以大大降低事务管理的复杂性。

**3、连接池的分配与释放**

　　连接池的分配与释放，对系统的性能有很大的影响。合理的分配与释放，可以提高连接的复用度，从而降低建立新连接的开销，同时还可以加快用户的访问速度。   
　　对于连接的管理可使用一个List。即把已经创建的连接都放入List中去统一管理。每当用户请求一个连接时，系统检查这个List中有没有可以分配的连接。如果有就把那个最合适的连接分配给他（如何能找到最合适的连接文章将在关键议题中指出）；如果没有就抛出一个异常给用户，List中连接是否可以被分配由一个线程来专门管理捎后我会介绍这个线程的具体实现。

**４、连接池的配置与维护**

　　连接池中到底应该放置多少连接，才能使系统的性能最佳？系统可采取设置最小连接数（minConnection）和最大连接数（maxConnection）等参数来控制连接池中的连接。比方说，最小连接数是系统启动时连接池所创建的连接数。如果创建过多，则系统启动就慢，但创建后系统的响应速度会很快；如果创建过少，则系统启动的很快，响应起来却慢。这样，可以在开发时，设置较小的最小连接数，开发起来会快，而在系统实际使用时设置较大的，因为这样对访问客户来说速度会快些。最大连接数是连接池中允许连接的最大数目，具体设置多少，要看系统的访问量，可通过软件需求上得到。   
　　如何确保连接池中的最小连接数呢？有动态和静态两种策略。动态即每隔一定时间就对连接池进行检测，如果发现连接数量小于最小连接数，则补充相应数量的新连接,以保证连接池的正常运转。静态是发现空闲连接不够时再去检查。

**数据库对比**

**第一、二代连接池**

区分一个数据库连接池是属于第一代产品还是代二代产品有一个最重要的特征就是看它在架构和设计时采用的线程模型，因为这直接影响的是并发环境下存取数据库连接的性能。

一般来讲采用单线程同步的架构设计都属于第一代连接池，二采用多线程异步架构的则属于第二代。比较有代表性的就是Apache Commons DBCP，在1.x版本中，一直延续着单线程设计模式，到2.x才采用多线程模型。

用版本发布时间来辨别区分两代产品，则一个偷懒的好方法。以下是这些常见数据库连接池最新版本的发布时间：

| **数据库连接池** | **最新版本** | **发布时间** |
| --- | --- | --- |
| C3P0 | c3p0-0.9.5.2 | on 9 Dec 2015 |
| DBCP | 2.2.0 | 27 December 2017 |
| Druid | 0.11.0 | Dec 4 2017 |
| HikariCP | 2.7.6 | 2018-01-14 |

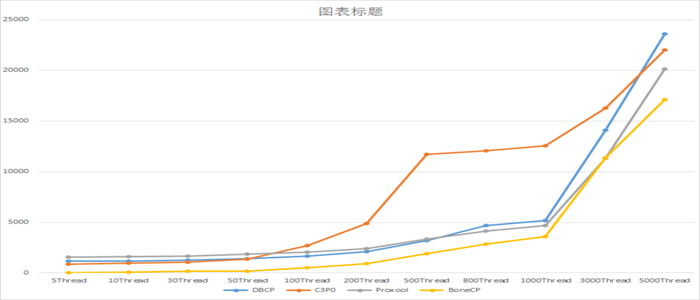
从表中可以看出，C3P0已经很久没有更新了。DBCP更新速度很慢，基本处于不活跃状态，而Druid和HikariCP处于活跃状态的更新中，这就是我们说的二代产品了。

二代产品对一代产品的超越是颠覆性的，除了一些“历史原因”，你很难再找到第二条理由说服自己不选择二代产品，但任何成功都不是偶然的，二代产品的成功很大程度上得益于前代产品们打下的基础，站在巨人的肩膀上，新一代的连接池的设计师们将这一项“工具化”的产品，推向了极致。其中，最具代表性的两款产品是：

* HikariCP
* Druid

**彻底死掉的C3P0**

C3P0是我使用的第一款数据库连接池，在很长一段时间内，它一直是Java领域内数据库连接池的代名词，当年盛极一时的Hibernate都将其作为内置的数据库连接池，可以业内对它的稳定性还是认可的。C3P0功能简单易用，稳定性好这是它的优点，但是性能上的缺点却让它彻底被打入冷宫。C3P0的性能很差，差到即便是同时代的产品相比它也是垫底的，更不用和Druid、HikariCP等相比了。正常来讲，有问题很正常，改就是了，但c3p0最致命的问题就是架构设计过于复杂，让重构变成了一项不可能完成的任务。随着国内互联网大潮的涌起，性能有硬伤的c3p0彻底的退出了历史舞台。

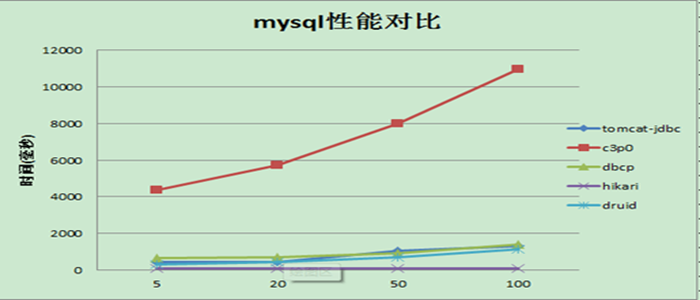


**咸鱼翻身的DBCP**

DBCP（DataBase Connection Pool）属于Apache顶级项目Commons中的核心子项目（最早在Jakarta Commons里就有）,在Apache的生态圈中的影响里十分广泛，比如最为大家所熟知的Tomcat就在内部集成了DBCP，实现JPA规范的OpenJPA，也是默认集成DBCP的。但DBCP并不是独立实现连接池功能的，它内部依赖于Commons中的另一个子项目Pool，连接池最核心的“池”，就是由Pool组件提供的，因此，DBCP的性能实际上就是Pool的性能，DBCP和Pool的依赖关系如下表：

| **Apache Commons DBCP** | **Apache Commons Pool** |
| --- | --- |
| v1.2.2 | v1.3 |
| v1.3 | v1.5.4 |
| v1.4 | v1.5.4 |
| v2.0.x | v2.2 |
| v2.1.x | v2.4.2 |
| v2.2.x | v2.5.0 |

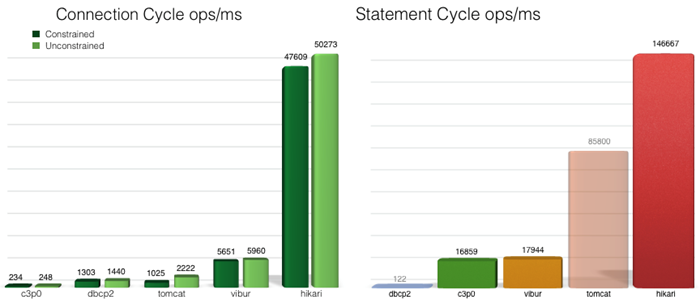
可以看到，因为核心功能依赖于Pool，所以DBCP本身只能做小版本的更新，真正大版本的更迭则完全依托于pool。有很长一段时间，pool都还是停留在1.x版本，这直接导致DBCP也更新乏力。很多依赖DBCP的应用在遇到性能瓶颈之后，别无选择，只能将其替换掉，DBCP忠实的拥趸tomcat就在其tomcat 7.0版本中，自己重新设计开发出了一套连接池（Tomcat JDBC Pool）。好在，在2013年事情终于迎来转机，13年9月Commons-Pool 2.0版本发布，14年2月份，DBCP也终于迎来了自己的2.0版本，基于新的线程模型全新设计的“池”让DBCP重焕青春，虽然和新一代的连接池相比仍有一定差距，但差距并不大，DBCP2.x版本已经稳稳达到了和新一代产品同级别的性能指标（见下图）。



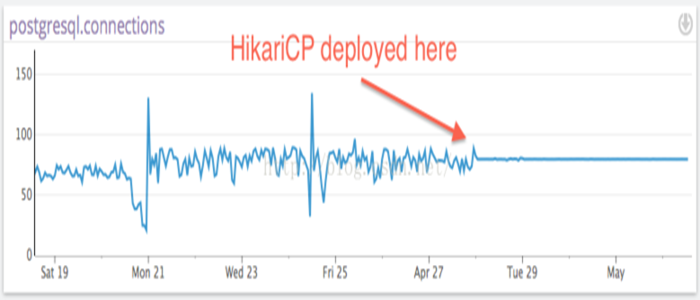
DBCP终于靠Pool咸鱼翻身，打了一个漂亮的翻身仗，但长时间的等待已经完全消磨了用户的耐心，与新一代的产品项目相比，DBCP没有任何优势，试问，谁会在有选择的前提下，去选择那个并不优秀的呢？也许，现在还选择DBCP2的唯一理由，就是情怀吧。

**性能无敌的HikariCP**

HikariCP号称“性能杀手”（It’s Faster），它的表现究竟如何呢，先来看下官网提供的数据：



不光性能强劲，稳定性也不差：



那它是怎么做到如此强劲的呢？官网给出的说明如下：

* 字节码精简：优化代码，直到编译后的字节码最少，这样，CPU缓存可以加载更多的程序代码；
* 优化代理和拦截器：减少代码，例如HikariCP的Statement proxy只有100行代码；
* 自定义数组类型（FastStatementList）代替ArrayList：避免每次get()调用都要进行range check，避免调用remove()时的从头到尾的扫描；
* 自定义集合类型（ConcurrentBag）：提高并发读写的效率；
* 其他缺陷的优化，比如对于耗时超过一个CPU时间片的方法调用的研究（但没说具体怎么优化）。

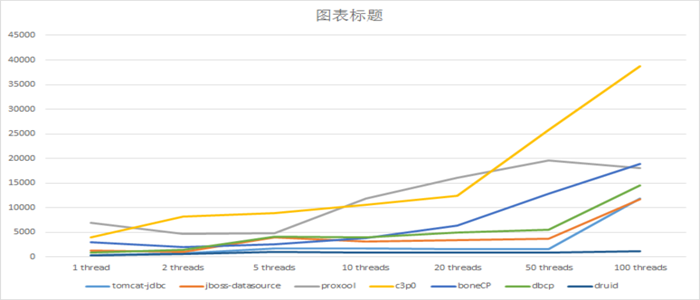
可以看到，上述这几点优化，和现在能找到的资料来看，HakariCP在性能上的优势应该是得到共识的，再加上它自身小巧的身形，在当前的“云时代、微服务”的背景下，HakariCP一定会得到更多人的青睐。

**功能全面的Druid**

近几年，阿里在开源项目上动作频频，除了有像fastJson、dubbo这类项目，更有像AliSQL这类的大型软件，今天说的Druid，就是阿里众多优秀开源项目中的一个。它除了提供性能卓越的连接池功能外，还集成了SQL监控，黑名单拦截等功能，用它自己的话说，Druid是“为监控而生”。借助于阿里这个平台的号召力，产品一经发布就赢得了大批用户的拥趸，从用户使用的反馈来看，Druid也确实没让用户失望。

相较于其他产品，Druid另一个比较大的优势，就是中文文档比较全面（毕竟是国人的项目么），在github的[wiki页面](https://github.com/alibaba/druid/wiki/%E9%A6%96%E9%A1%B5)，列举了日常使用中可能遇到的问题，对一个新用户来讲，上面提供的内容已经足够指导它完成产品的配置和使用了。

下图为Druid自己提供的性能测试数据：



现在项目开发中，我还是比较倾向于使用Durid，它不仅仅是一个数据库连接池，它还包含一个ProxyDriver，一系列内置的JDBC组件库，一个SQL Parser。

**Druid 相对于其他数据库连接池的优点**

1. 强大的监控特性，通过Druid提供的监控功能，可以清楚知道连接池和SQL的工作情况。

a. 监控SQL的执行时间、ResultSet持有时间、返回行数、更新行数、错误次数、错误堆栈信息；

b. SQL执行的耗时区间分布。什么是耗时区间分布呢？比如说，某个SQL执行了1000次，其中0~1毫秒区间50次，1~10毫秒800次，10~100毫秒100次，100~1000毫秒30次，1~10秒15次，10秒以上5次。通过耗时区间分布，能够非常清楚知道SQL的执行耗时情况；

c. 监控连接池的物理连接创建和销毁次数、逻辑连接的申请和关闭次数、非空等待次数、PSCache命中率等。

1. 方便扩展。Druid提供了Filter-Chain模式的扩展API，可以自己编写Filter拦截JDBC中的任何方法，可以在上面做任何事情，比如说性能监控、SQL审计、用户名密码加密、日志等等。
2. Druid集合了开源和商业数据库连接池的优秀特性，并结合阿里巴巴大规模苛刻生产环境的使用经验进行优化。

**总结**

时至今日，虽然每个应用（需要RDBMS的）都离不开连接池，但在实际使用的时候，连接池已经可以做到“隐形”了。也就是说在通常情况下，连接池完成项目初始化配置之后，就再不需要再做任何改动了。不论你是选择Druid或是HikariCP，甚至是DBCP，它们都足够稳定且高效！之前讨论了很多关于连接池的性能的问题，但这些性能上的差异，是相较于其他连接池而言的，对整个系统应用来说，第二代连接池在使用过程中体会到的差别是微乎其微的，基本上不存在因为连接池的自身的配饰和使用导致系统性能下降的情况，除非是在单点应用的数据库负载足够高的时候（压力测试的时候），但即便是如此，通用的优化的方式也是单点改集群，而不是在单点的连接池上死扣。