第九周总结

首先是听课笔记：

9.1 数据库的基本原理

1. 数据库基本架构

    1. 数据库架构：SQL -> 连接器 -> 语法分析器 -> 语义分析与优化器 -> 执行计划 -> 执行引擎

    2. 连接器：数据库会为连接分配一块专门的内存进行上下文会话管理，不同于网络连接，数据库的连接更重一些，需要花费更多时间。一般数据库连接会在初始化的时候建立好。

    3. 语法分析器：SQL -> 抽象语法树

    4. 语义分析与优化：将各种复杂嵌套的SQL进行等价语义转化，得到有限的几种关系代数计算结构，并利用索引等信息进一步优化。不同数据库性能的差别主要就在这里。

    5. 执行计划：就是我们用explain的时候看到的那个

2. PrepareStatement预编译：预编译可以提前执行到生成执行计划这一步，这有2个优点

    1. 如果对预编译结果进行缓存，可以提高执行效率

    2. 可以防止sql注入攻击

3. 数据存储：

    1. B+树：数据库以B+树的形式存储，数据在叶子节点上，一般叶子节点也会简单进行连接构成一个链表。

    2. 聚簇索引：数据和索引放在一起，数据在叶子上，这既是一颗索引树也是一颗数据树。一般主键索引就是聚簇索引。

    3. 非聚簇索引：叶子节点所存的是聚簇索引，而非数据。也就是说，在别的字段建立索引时，要先通过那个字段的索引找到主键，然后再通过主键索引找到目标数据

4. 合理使用索引

    1. 生产环境中用alter操作要谨慎，因为会阻塞数据库的增删改操作，且消耗时间长，分钟级，而且由于连接无法释放，查询大概率也会被阻塞

    2. 删除不必要的索引，避免不必要的开销

    3. 使用更小的数据类型去创建索引

5. 数据库事务：ACID

    1. 数据库事务日志中记录着所有的undo信息和redo信息，由事务决定最后是undo还是redo，从而保持数据库操作的一致性

9.2 JVM虚拟机架构原理

1. JVM组成架构：JVM本身是一个进程，它内部有自己的类加载器，运行期数据区，以及执行引擎，看起来就像是一个机器，所以叫虚拟机。

2. 字节码：Java总共有200多个指令，所有指令都可以用一个字节表示，所以又叫字节码。命令执行过程中需要指令和对应的操作数

3. 字节码文件编译过程：Java源文件->词法解析->token流->语法解析->语义分析->生成字节码

4. 自定义类加载器：

    1. 隔离加载类：同一个JVM中不同组件加载同一个类的不同版本

    2. 扩展加载类：从网络，数据库等处加载字节码

    3. 字节码加密：加载自定义的加密字节码，在classloader中解密

5. 堆和栈：

    1. 堆：一个JVM进程只有一个堆，所有JVM线程共享

    2. 堆栈：每个线程有自己的堆栈。创建一个对象时，会在堆中创建，并在线程的栈上保留引用。

    3. 方法区和程序计数器：方法区是从磁盘读进来的类字节码，程序计数器则指向执行的代码，记录执行进度。

6. 线程工作内存和volatile：Java每个线程都有自己的CPU缓存，即线程的工作内存，这个内存里面的值改变不一定会立即写回到主内存，导致其他的线程读到脏数据。volatile就是用来保证这个数据对各个线程都是可见的。

9.3 JVM垃圾回收性能分析：

1. JVM的垃圾回收：

    1. 标记：通过可达性分析找出可以被清理的对象并回收

    2. 清理：把没用的对象进行删除

    3. 压缩：把剩下有用的对象压缩到一块连续的内存去

    4. 复制：把存活对象复制到一块连续的内存去

2. JVM的分代垃圾回收：用来缩小平均每次回收检索的内存范围，从而提高回收效率

3. 垃圾回收器实现：串行回收器，并行回收器，并发回收器CMS，G1回收器

4. JVM启动参数：标准参数、非标准参数

5. 运行分析相关的命令：JPS，JSTAT，JSTACK，JConsole，JVisualVM

9.4 java代码优化技巧及原理

1. 合理并谨慎使用多线程：启动线程数=（任务执行时间/（任务执行时间-阻塞等待时间））*CPU内核数

2. 竞态条件与临界区：多线程必须获得锁才能进入临界区

3. java线程安全。局部变量和局部对象引用：一个变量对象仅在方法内部被创建和使用则是线程安全的；对象成员变量：一个对象在堆中创建，并且成员变量被共享则是不安全的。

4. threadLocal，本质是一个静态的map，key是thread，value是每个thread自己的map，每个thread在进行set和get的时候只操作自己对应的那个map

5. java内存泄漏：逻辑上的内存泄漏。如长生命周期对象，静态容器，缓存等

6. 合理使用池化对象

7. 合理使用jdk容器，对性能影响很大

8. 使用IO buffer和NIO：异步无阻塞IO通信，相对来说性能更好

9. 使用组合代替继承：减少耦合；避免继承太深导致创建对象时损失性能

10. 合理使用单例模式。尽量无状态，性能安全。

9.5 系统性能优化案例：秒杀系统（上）

1. 一个秒杀活动，5天时间，100倍并发。能搞定这件事，算是人生机会....李老师这课程很实在- -

2. 还有机器人并发秒杀，ddos攻击，以及提前进入下单页面等问题

9.6 系统性能优化案例：秒杀系统（二）

1. 相对来说开销比较小的方案是开发一套新系统专用于秒杀

2. 带宽准备：出口从1G上升到2.5G，图片等静态资源依赖cdn

3. 新系统仅有三个页面：秒杀商品列表，商品介绍，订单填写，其中支付可以用原来的，因为能够成功进入支付的人很少

4. 性能方面：

    1. 静态化：将原来的动态页面转化为静态页面，不走搜索系统和数据库最大化利用CDN

    2. 并发控制，防秒杀器：设置阀门，只放最前面的一部分人进入系统。

    3. 简化流程：砍掉不重要的分支，如下单页的所有数据库查询；以下单作为秒杀成的标志，支付在一天内完成即可。

    4. 前端优化：采用YSLOW原则提升响应速度

5. 控制客户端跳过秒杀页直接进下单页：目标url不缓存，每次都访问后端，活动开始后才真正返回有效url

6. 三级阀门计数限制：只有1000人能进秒杀页面，只有100人能进填写订单页，只有56人能进支付页，越往后，并发越小

个人总结

这一周讲的内容也比较多，主要包括数据库，JVM，Java代码本身的优化，以及一个秒杀系统的实例。

其中，数据库方面主要是剖析了其架构，访问的过程，并着重强调了索引对于数据库访问的优化以及重要性。

JVM虚拟机主要讲了JVM的基本架构垃圾回收的原理和性能调优等。

Java代码方面主要是线程安全相关的知识，以及一些优秀的实践，其中，ThreadLocal的实现让人眼前一亮，值得学习一下。

最后的秒杀系统用意想不到的简单的方法解决了一个非常大的问题，可以说是智慧的集中体现了，这也说明了架构师思考能力的重要性以及意义之所在。