

二

36 谈谈MySQL支持的事务隔离级别，以及悲观锁和乐观锁的原理和应用场景？-极客时间

在日常开发中，尤其是业务开发，少不了利用 Java 对数据库进行基本的增删改查等数据操作，这也是 Java 工程师的必备技能之一。做好数据操作，不仅仅需要对 Java 语言相关框架的掌握，更需要对各种数据库自身体系结构的理解。今天这一讲，作为补充 Java 面试考察知识点的完整性，关于数据库的应用和细节还需要在实践中深入学习。

今天我要问你的问题是，**谈谈 MySQL 支持的事务隔离级别，以及悲观锁和乐观锁的原理和应用场景？**

典型回答

所谓隔离级别 ([Isolation Level](#))，就是在数据库事务中，为保证并发数据读写的正确性而提出的定义，它并不是 MySQL 专有的概念，而是源于[ANSI/ISO](#)制定的[SQL-92](#)标准。

每种关系型数据库都提供了各自特色的隔离级别实现，虽然在通常的[定义](#)中是以锁为实现单元，但实际的实现千差万别。以最常见的 MySQL InnoDB 引擎为例，它是基于 [MVCC](#) (Multi-Versioning Concurrency Control) 和锁的复合实现，按照隔离程度从低到高，MySQL 事务隔离级别分为四个不同层次：

- 读未提交 (Read uncommitted)，就是一个事务能够看到其他事务尚未提交的修改，这是最低的隔离水平，允许[脏读](#)出现。
- 读已提交 (Read committed)，事务能够看到的数据都是其他事务已经提交的修改，也就是保证不会看到任何中间性状态，当然脏读也不会出现。读已提交仍然是比较低级别的隔离，并不保证再次读取时能够获取同样的数据，也就是允许其他事务并发修改数据，允许不可重复读和幻象读 (Phantom Read) 出现。
- 可重复读 (Repeatable reads)，保证同一个事务中多次读取的数据是一致的，这是 MySQL InnoDB 引擎的默认隔离级别，但是和一些其他数据库实现不同的是，可以简单认为 MySQL 在可重复读级别不会出现幻象读。
- 串行化 (Serializable)，并发事务之间是串行化的，通常意味着读取需要获取共享读锁，更新需要获取排他写锁，如果 SQL 使用 WHERE 语句，还会获取区间锁 (MySQL

以 GAP 锁形式实现，可重复读级别中默认也会使用），这是最高的隔离级别。

至于悲观锁和乐观锁，也并不是 MySQL 或者数据库中独有的概念，而是并发编程的基本概念。主要区别在于，操作共享数据时，“悲观锁”即认为数据出现冲突的可能性更大，而“乐观锁”则是认为大部分情况不会出现冲突，进而决定是否采取排他性措施。

反映到 MySQL 数据库应用开发中，悲观锁一般就是利用类似 `SELECT ... FOR UPDATE` 这样的语句，对数据加锁，避免其他事务意外修改数据。乐观锁则与 Java 并发包中的 `AtomicFieldUpdater` 类似，也是利用 CAS 机制，并不会对数据加锁，而是通过对比数据的时间戳或者版本号，来实现乐观锁需要的版本判断。

我认为前面提到的 MVCC，其本质就可以看作是种乐观锁机制，而排他性的读写锁、双阶段锁等则是悲观锁的实现。

有关它们的应用场景，你可以构建一下简化的火车余票查询和购票系统。同时查询的人可能很多，虽然具体座位票只能是卖给一个人，但余票可能很多，而且也并不能预测哪个查询者会购票，这个时候就更适合用乐观锁。

考点分析

今天的问题来源于实际面试，这两部分问题反映了面试官试图考察面试者在日常应用开发中，是否学习或者思考过数据库内部的机制，是否了解并发相关的基础概念和实践。

我从普通数据库应用开发者的角度，提供了一个相对简化的答案，面试官很有可能进一步从实例的角度展开，例如设计一个典型场景重现脏读、幻象读，或者从数据库设计的角度，可以用哪些手段避免类似情况。我建议你在准备面试时，可以在典型的数据库上试验一下，验证自己的观点。

其他可以考察的点也有很多，在准备这个问题时你也可以对比 Java 语言的并发机制，进行深入理解，例如，随着隔离级别从低到高，竞争性（Contention）逐渐增强，随之而来的代价同样是性能和扩展性的下降。

数据库衍生出很多不同的职责方向：

- 数据库管理员（DBA），这是一个单独的专业领域。
- 数据库应用工程师，很多业务开发者就是这种定位，综合利用数据库和其他编程语言等技能，开发业务应用。
- 数据库工程师，更加侧重于开发数据库、数据库中间件等基础软件。

后面两者与 Java 开发更加相关，但是需要的知识和技能是不同的，所以面试的考察角度也

有区别，今天我会分析下对相关知识学习和准备面试的看法。

另外，在数据库相关领域，Java 工程师最常接触到的就是 O/R Mapping 框架或者类似的数据库交互类库，我会选取最广泛使用的框架进行对比和分析。

知识扩展

首先，我来谈谈对数据库相关领域学习的看法，从最广泛的应用开发者角度，至少需要掌握：

- 数据库设计基础，包括数据库设计中的几个基本范式，各种数据库的基础概念，例如表、视图、索引、外键、序列号生成器等，清楚如何将现实中业务实体和其依赖关系映射到数据库结构中，掌握典型实体数据应该使用什么样的数据库数据类型等。
- 每种数据库的设计和实现多少会存在差异，所以至少要精通你使用过的数据库的设计要点。我今天开篇谈到的 MySQL 事务隔离级别，就区别于其他数据库，进一步了解 MVCC、Locking 等机制对于处理进阶问题非常有帮助；还需要了解，不同索引类型的使用，甚至是底层数据结构和算法等。
- 常见的 SQL 语句，掌握基础的 SQL 调优技巧，至少要了解基本思路是怎样的，例如 SQL 怎样写才能更好利用索引、知道如何分析 SQL 执行计划等。
- 更进一步，至少需要了解针对高并发等特定场景中的解决方案，例如读写分离、分库分表，或者如何利用缓存机制等，目前的数据存储也远不止传统的关系型数据库了。



上面的示意图简单总结了我对数据库领域的理解，希望可以给你进行准备时提供个借鉴。当然在准备面试时并不是一味找一堆书闷头苦读，我还是建议从实际工作中使用的数据库出发，侧重于结合实践，完善和深化自己的知识体系。

接下来我们还是回到 Java 本身，目前最为通用的 Java 和数据库交互技术就是 JDBC，最常见的开源框架基本都是构建在 JDBC 之上，包括我们熟悉的JPA/Hibernate、MyBatis、Spring JDBC Template 等，各自都有独特的设计特点。

Hibernate 是最负盛名的 O/R Mapping 框架之一，它也是一个 JPA Provider。顾名思义，它是以对象为中心的，其强项更体现在数据库到 Java 对象的映射，可以很方便地在 Java 对象层面体现外键约束等相对复杂的关系，提供了强大的持久化功能。内部大量使用了Lazy-load等技术提高效率。并且，为了屏蔽数据库的差异，降低维护开销，Hibernate 提供了类 SQL 的 HQL，可以自动生成某种数据库特定的 SQL 语句。

Hibernate 应用非常广泛，但是过度强调持久化和隔离数据库底层细节，也导致了很多弊端，例如 HQL 需要额外的学习，未必比深入学习 SQL 语言更高效；减弱程序员对 SQL 的直接控制，还可能导致其他代价，本来一句 SQL 的事情，可能被 Hibernate 生成几条，隐藏的内部细节也阻碍了进一步的优化。

而 MyBatis 虽然仍然提供了一些映射的功能，但更加以 SQL 为中心，开发者可以侧重于 SQL 和存储过程，非常简单、直接。如果我们的应用需要大量高性能的或者复杂的 SELECT 语句等，“半自动”的 MyBatis 就会比 Hibernate 更加实用。

而 Spring JDBC Template 也是更加接近于 SQL 层面，Spring 本身也可以集成 Hibernate 等 O/R Mapping 框架。

关于这些具体开源框架的学习，我的建议是：

- 从整体上把握主流框架的架构和设计理念，掌握主要流程，例如 SQL 解析生成、SQL 执行到结果映射等处理过程到底发生了什么。
- 掌握映射等部分的细节定义和原理，根据我在准备专栏时整理的面试题目，发现很多题目都是偏向于映射定义的细节。
- 另外，对比不同框架的设计和实现，既有利于你加深理解，也是面试考察的热点方向之一。

今天我从数据库应用开发者的角度，分析了 MySQL 数据库的部分内部机制，并且补充了我对数据库相关面试准备和知识学习的建议，最后对主流 O/R Mapping 等框架进行了简单的对比。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？今天的思考题是，从架构设计的角度，可以将 MyBatis 分为哪几层？每层都有哪些主要模块？

请你在留言区写写你对这个问题的思考，我会选出经过认真思考的留言，送给你一份学习奖励礼券，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。

[上一页](#)

[下一页](#)