

大家好呀！今天和小伙伴聊聊读写分离以及分库分表。

相信很多小伙伴们对于这两个概念已经比较熟悉了，这篇文章全程都是大白话的形式，希望能够给你带来不一样的感受。

如果你之前不太了解这两个概念，那我建议你搞懂之后，可以把自己对于读写分离以及分库分表的理解讲给你的同事/朋友听听。

原创不易，若有帮助，点赞/分享就是对我最大的鼓励！

个人能力有限。如果文章有任何需要补充/完善/修改的地方，欢迎在评论区指出，共同进步！

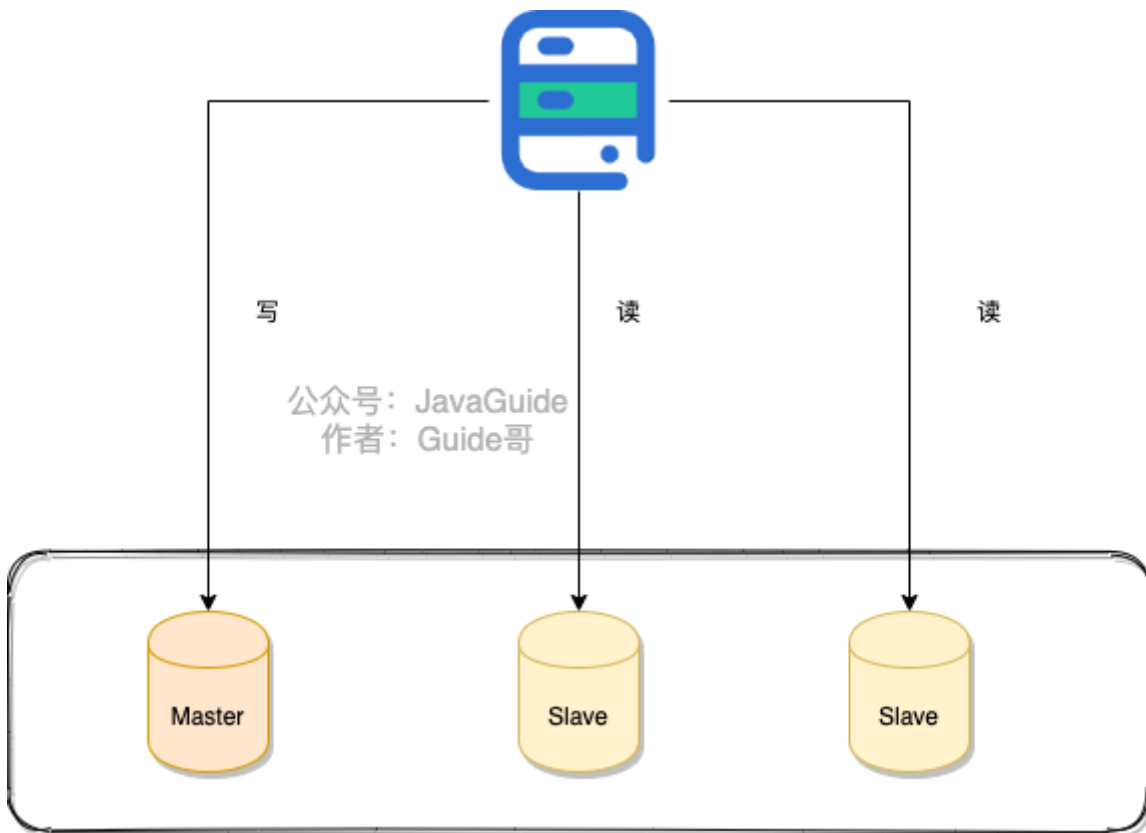
读写分离&分库分表

读写分离

何为读写分离？

见名思意，根据读写分离的名字，我们就可以知道：**读写分离主要是为了将对数据库的读写操作分散到不同的数据库节点上**。这样的话，就能够小幅提升写性能，大幅提升读性能。

我简单画了一张图来帮助不太清楚读写分离的小伙伴理解。



一般情况下，我们都会选择一主多从，也就是一台主数据库负责写，其他的从数据库负责读。主库和从库之间会进行数据同步，以保证从库中数据的准确性。这样的架构实现起来比较简单，并且也符合系统的写少读多的特点。

读写分离会带来什么问题？如何解决？

读写分离对于提升数据库的并发非常有效，但是，同时也会引来一个问题：主库和从库的数据存在延迟，比如你写完主库之后，主库的数据同步到从库是需要时间的，这个时间差就导致了主库和从库的数据不一致性问题。这也就是我们经常说的 **主从同步延迟**。

主从同步延迟问题的解决，没有特别好的一种方案（可能是我太菜了，欢迎评论区补充）。你可以根据自己的业务场景，参考一下下面几种解决办法。

1.强制将读请求路由到主库处理。

既然你从库的数据过期了，那我就直接从主库读取嘛！这种方案虽然会增加主库的压力，但是，实现起来比较简单，也是我了解到的使用最多的一种方式。

比如 **Sharding-JDBC** 就是采用的这种方案。通过使用 Sharding-JDBC 的 **HintManager** 分片键值管理器，我们可以强制使用主库。

```
HintManager hintManager = HintManager.getInstance();
hintManager.setMasterRouteOnly();
// 继续JDBC操作
```

对于这种方案，你可以将那些必须获取最新数据的读请求都交给主库处理。

2.延迟读取。

还有一些朋友肯定会想既然主从同步存在延迟，那我就在延迟之后读取啊，比如主从同步延迟 0.5s,那我就 1s 之后再读取数据。这样多方便啊！方便是方便，但是也很扯淡。

不过，如果你是这样设计业务流程就会好很多：对于一些对数据比较敏感的场景，你可以在完成写请求之后，避免立即进行请求操作。比如你支付成功之后，跳转到一个支付成功的页面，当你点击返回之后才返回自己的账户。

另外，《MySQL 实战 45 讲》这个专栏中的《读写分离有哪些坑？》这篇文章还介绍了很多其他比较实际的解决办法，感兴趣的小伙伴可以自行研究一下。

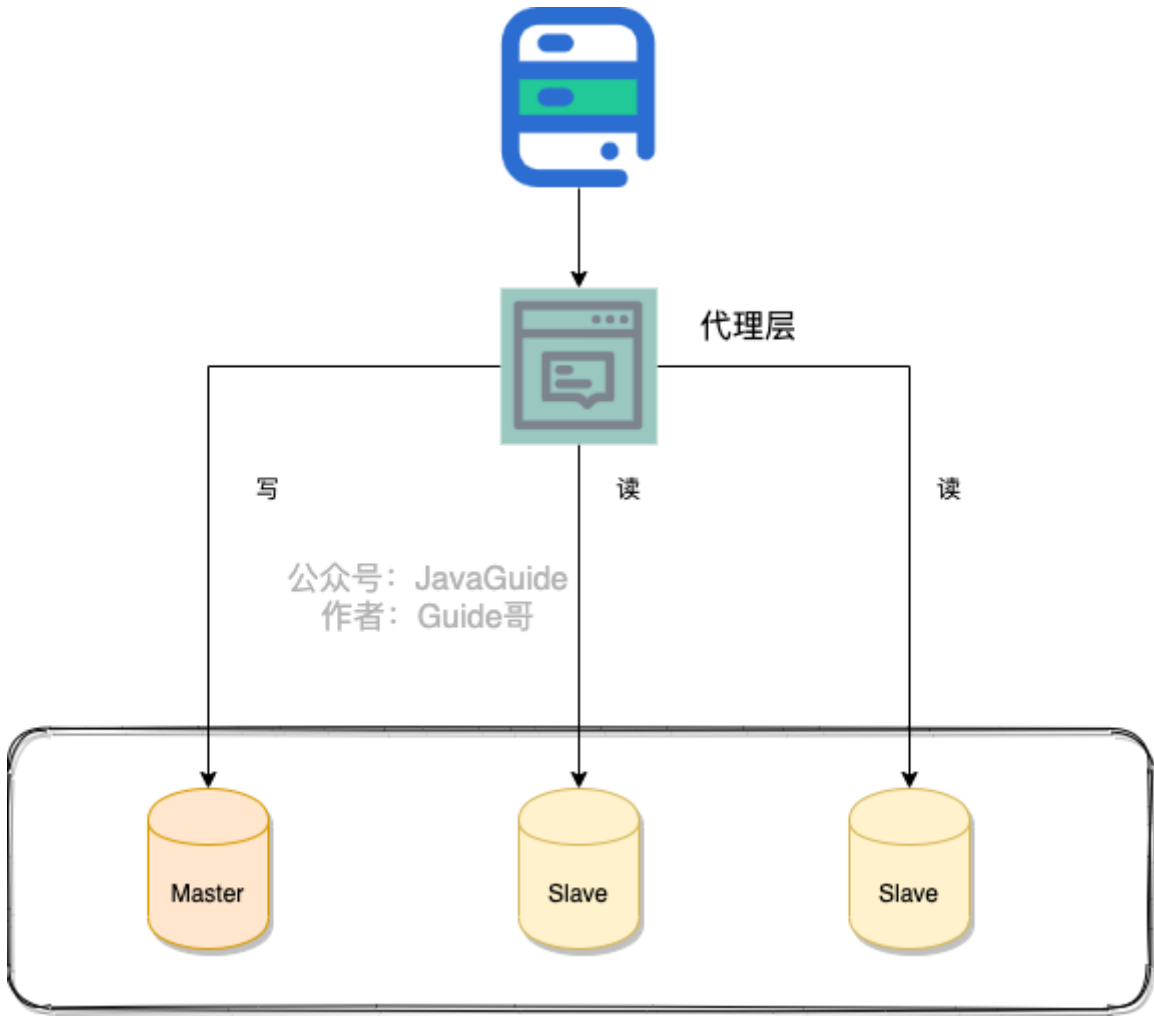
如何实现读写分离？

不论是使用哪一种读写分离具体的实现方案，想要实现读写分离一般包含如下几步：

1. 部署多台数据库，选择一种的一台作为主数据库，其他的一台或者多台作为从数据库。
2. 保证主数据库和从数据库之间的数据是实时同步的，这个过程也就是我们常说的**主从复制**。
3. 系统将写请求交给主数据库处理，读请求交给从数据库处理。

落实到项目本身的话，常用的方式有两种：

1.代理方式



我们可以在应用和数据中间加了一个代理层。应用程序所有的数据请求都交给代理层处理，代理层负责分离读写请求，将它们路由到对应的数据库中。

提供类似功能的中间件有 **MySQL Router**（官方）、**Atlas**（基于 MySQL Proxy）、**Maxscale**、**MyCat**。

2.组件方式

在这种方式中，我们可以通过引入第三方组件来帮助我们读写请求。

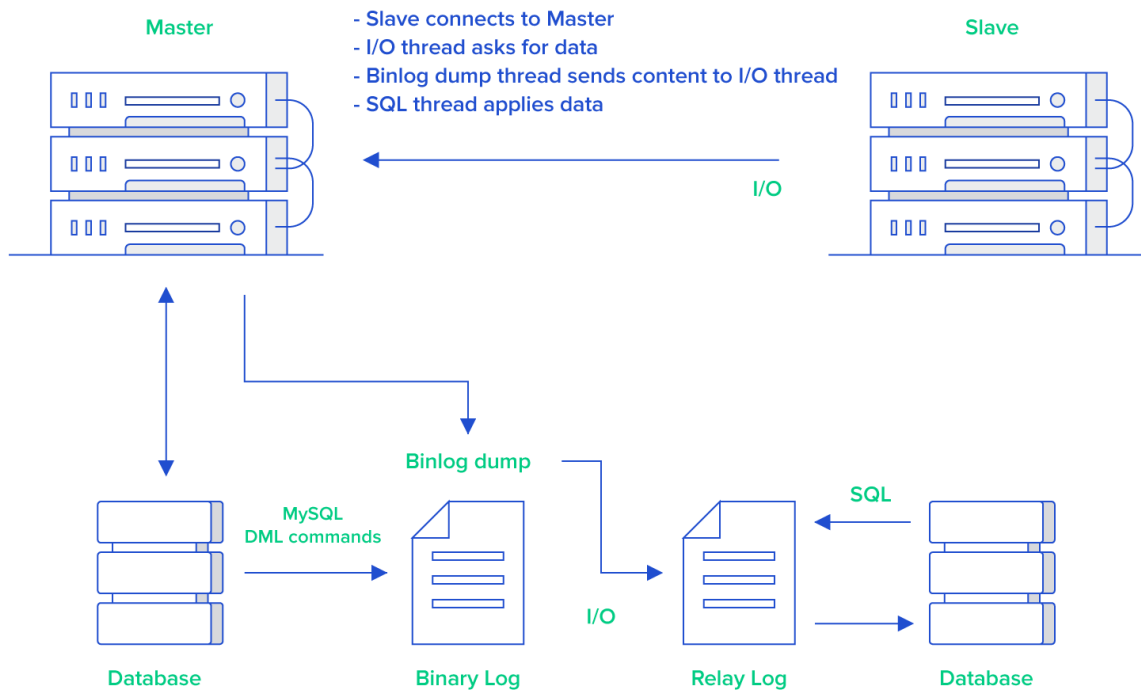
这也是我比较推荐的一种方式。这种方式目前在各种互联网公司中用的最多的，相关的实际的案例也非常多。如果你要采用这种方式的话，推荐使用 **sharding-jdbc**，直接引入 jar 包即可使用，非常方便。同时，也节省了很多运维的成本。

你可以在 **shardingsphere** 官方找到[sharding-jdbc 关于读写分离的操作] (<https://shardingsphere.apache.org/document/legacy/3.x/document/cn/manual/sharding-jdbc/usage/read-write-splitting/>)。

主从复制原理了解么？

MySQL binlog(binary log 即二进制日志文件) 主要记录了 MySQL 数据库中数据的所有变化(数据库执行的所有 DDL 和 DML 语句)。因此，我们根据主库的 MySQL binlog 日志就能够将主库的数据同步到从库中。


更具体和详细的过程是这个样子的（图片来自于：《[MySQL Master-Slave Replication on the Same Machine](#)》）：



1. 主库将数据库中数据的变化写入到 binlog
2. 从库连接主库
3. 从库会创建一个 I/O 线程向主库请求更新的 binlog
4. 主库会创建一个 binlog dump 线程来发送 binlog，从库中的 I/O 线程负责接收
5. 从库的 I/O 线程将接收的 binlog 写入到 relay log 中。
6. 从库的 SQL 线程读取 relay log 同步数据本地（也就是再执行一遍 SQL）。


怎么样？看了我对主从复制这个过程的讲解，你应该搞明白了吧！

你一般看到 binlog 就要想到主从复制。当然，除了主从复制之外，binlog 还能帮助我们实现数据恢复。

 拓展一下：

不知道大家有没有使用过阿里开源的一个叫做 canal 的工具。这个工具可以帮助我们实现 MySQL 和其他数据源比如 Elasticsearch 或者另外一台 MySQL 数据库之间的数据同步。很显然，这个工具的底层原理肯定也是依赖 binlog。canal 的原理就是模拟 MySQL 主从复制的过程，解析 binlog 将数据同步到其他的数据源。

另外，像咱们常用的分布式缓存组件 Redis 也是通过主从复制实现的读写分离。

 简单总结一下：

MySQL 主从复制是依赖于 binlog。另外，常见的一些同步 MySQL 数据到其他数据源的工具（比如 canal）的底层一般也是依赖 binlog。

分库分表

读写分离主要应对的是数据库读并发，没有解决数据库存储问题。试想一下：**如果 MySQL 一张表的数据量过大怎么办？**

换言之，**我们该如何解决 MySQL 的存储压力呢？**

答案之一就是 **分库分表**。

何为分库？

分库 就是将数据库中的数据分散到不同的数据库上。

下面这些操作都涉及到了分库：

- 你将数据库中的用户表和用户订单表分别放在两个不同的数据库。
- 由于用户表数据量太大，你对用户表进行了水平切分，然后将切分后的 2 张用户表分别放在两个不同的数据库。

何为分表？

分表 就是对单表的数据进行拆分，可以是垂直拆分，也可以是水平拆分。

何为垂直拆分？

简单来说，垂直拆分是对数据表列的拆分，把一张列比较多的表拆分为多张表。

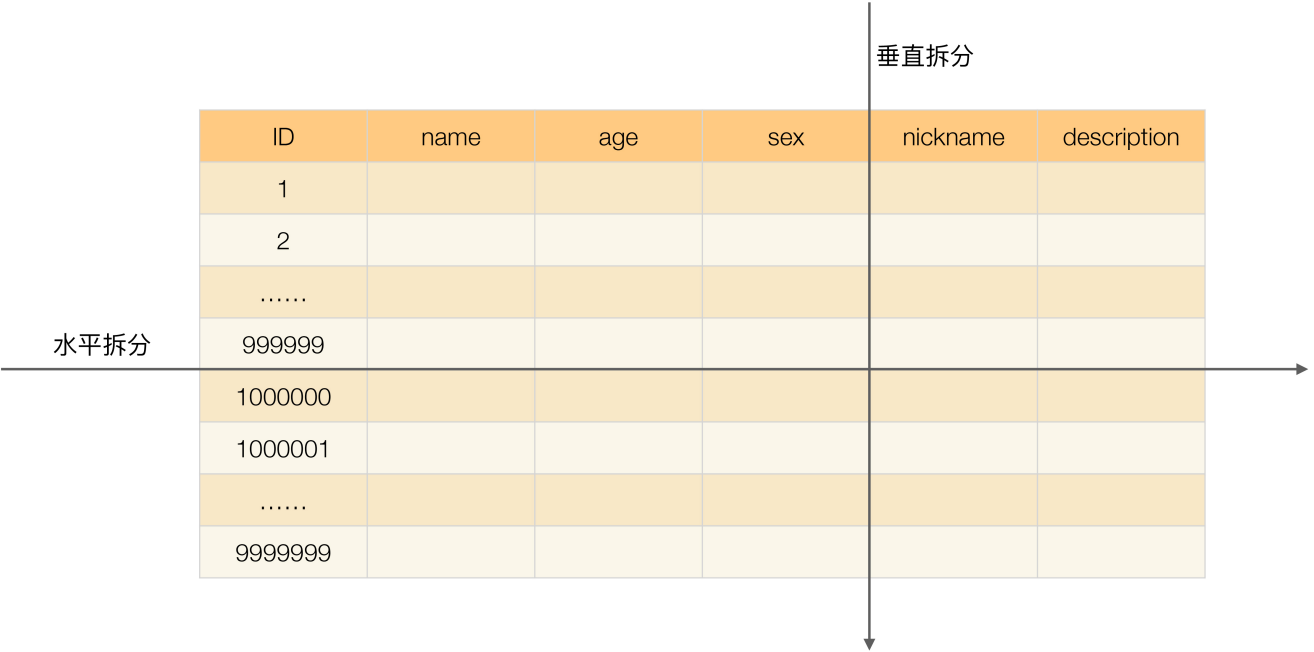
举个例子：我们可以将用户信息表中的一些列单独抽出来作为一个表。

何为水平拆分？

简单来说，水平拆分是对数据表行的拆分，把一张行比较多的表拆分为多张表。

举个例子：我们可以将用户信息表拆分成多个用户信息表，这样就可以避免单一表数据量过大对性能造成影响。

[《从零开始学架构》](#) 中的有一张图片对于垂直拆分和水平拆分的描述还挺直观的。



什么情况下需要分库分表？

遇到下面几种场景可以考虑分库分表：

- 单表的数据达到千万级别以上，数据库读写速度比较缓慢（分表）。
- 数据库中的数据占用的空间越来越大，备份时间越来越长（分库）。
- 应用的并发量太大（分库）。

分库分表会带来什么问题呢？

记住，你在公司做的任何技术决策，不光是要考虑这个技术能不能满足我们的要求，是否适合当前业务场景，还要重点考虑其带来的成本。

引入分库分表之后，会给系统带来什么挑战呢？

- **join 操作**： 同一个数据库中的表分布在了不同的数据库中，导致无法使用 join 操作。这样就导致我们需要手动进行数据的封装，比如你在一个数据库中查询到一个数据之后，再根据这个数据去另外一个数据库中找到对应的数据。
- **事务问题**： 同一个数据库中的表分布在了不同的数据库中，如果单个操作涉及到多个数据库，那么数据库自带的事务就无法满足我们的要求了。
- **分布式 id**： 分库之后，数据遍布在不同服务器上的数据库，数据库的自增主键已经没办法满足生成的主键唯一了。我们如何为不同的数据节点生成全局唯一主键呢？这个时候，我们就需要为我们的系统引入分布式 id 了。
-

另外，引入分库分表之后，一般需要 DBA 的参与，同时还需要更多的数据库服务器，这些都属于成本。

分库分表有没有什么比较推荐的方案？

ShardingSphere 项目（包括 Sharding-JDBC、Sharding-Proxy 和 Sharding-Sidecar）是当当捐入 Apache 的，目前主要由京东数科的一些巨佬维护。

Apache ShardingSphere核心功能



ShardingSphere 绝对可以说是当前分库分表的首选！ShardingSphere 的功能完善，除了支持读写分离和分库分表，还提供分布式事务、数据库治理等功能。

另外，ShardingSphere 的生态体系完善，社区活跃，文档完善，更新和发布比较频繁。

芴芴之前写了一篇分库分表的实战文章，各位朋友可以看看：[《芋道 Spring Boot 分库分表入门》](#)。

分库分表后，数据怎么迁移呢？

分库分表之后，我们如何将老库（单库单表）的数据迁移到新库（分库分表后的数据库系统）呢？

比较简单同时也是非常常用的方案就是**停机迁移**，写个脚本老库的数据写到新库中。比如你在凌晨 2 点，系统使用的人数非常少的时候，挂一个公告说系统要维护升级预计 1 小时。然后，你写一个脚本将老库的数据都同步到新库中。

如果你不想停机迁移数据的话，也可以考虑**双写方案**。双写方案是针对那种不能停机迁移的场景，实现起来要稍微麻烦一些。具体原理是这样的：

- 我们对老库的更新操作（增删改），同时也要写入新库（双写）。如果操作的数据不存在于新库的话，需要插入到新库中。这样就能保证，咱们新库里的数据是最新的。
- 在迁移过程，双写只会让被更新操作过的老库中的数据同步到新库，我们还需要自己写脚本将老库中的数据和新的数据做比对。如果新库中没有，那咱们就把数据插入到新库。如果新库有，旧库没有，就把新库对应的数据删除（冗余数据清理）。
- 重复上一步的操作，直到老库和新库的数据一致为止。

想要在项目中实施双写还是比较麻烦的，很容易会出现问题。我们可以借助上面提到的数据库同步工具 Canal 做增量数据迁移（还是依赖 binlog，开发和维护成本较低）。

