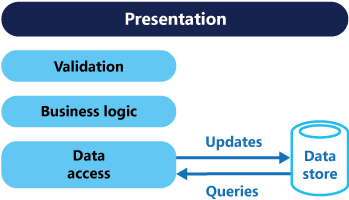
<https://www.cnblogs.com/yangecnu/p/Introduction-CQRS.html>

随着系统逻辑变得复杂，用户增多，这些设计就会出现一些性能问题，虽然再DB上可以做一些读写分离的设计，但在业务上如果在读写方面混合在一起，仍然会出现一些问题，

通常对DB执行的增（Create），查（Retrieve），改（Update），删（Delete）都是针对的系统的实体对象，如通过数据访问层获取数据，然后通过数据传输对象DTO传给表现层。或者，用户需要更新数据，通过DTO对象将数据传给Model，然后通过数据访问层写回数据库，系统中的所有交互都是和数据查询和存储有关，可以认为是数据驱动（Data-Driven）

对于一些比较简单的系统，使用CRUD的设计方式能够满足要求。特别是通过一些代码生成工具及ORM等能够非常方便快速的实现功能

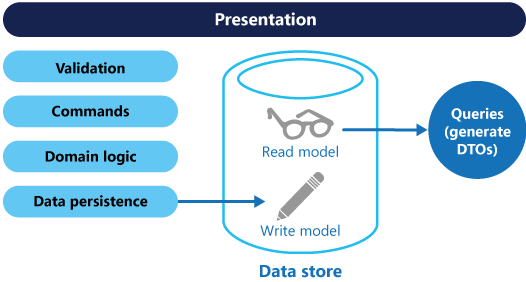
缺点：1.使用同一个对象实体来进行数据库读写可能会太粗糙，因为大多数情况下只需要操作部分字段或者个别 2.使用同一实体对象对同一数据进行读写操作的时候，可能会遇到资源竞争的情况，经常要处理的锁的问题，在写入数据的时候，需要加锁（？ 哪种锁，简单数据库锁吗？CQRS中如何避免的）。读取数据的时候判断是否允许脏读（？什么是脏读），这样会使得系统逻辑性和复杂性增加，对系统吞吐量的增长会产生影响 3.同步的，直接与数据库进行交互在大数据量同时访问的情况下可能会影响性能和响应性，并且可能会产生性能瓶颈 4. 由于同一实体对象都会在读写操作中用到，所以对于安全和权限的管理会变得比较复杂



命令查询职责分离模式（Command Query Responsibility Segregation ，CQRS），该模式从业务上分离修改（Command，增，删，改，会对系统状态进行修改）和查询（Query，查，不会对系统状态进行修改）的行为。从而使得逻辑更加清晰，便于对不同部分进行针对性的优化，这种分离是基于方法是执行命令还是执行查询这一原则来定的

CQRS使用分离的接口将数据查询操作（Queries）和数据修改操作（Commands）分离出来，这也意味着在查询和更新过程中使用的数据模型也是不一样的。这样读写逻辑就隔离出来了。

CQS



Validation：验证

Commands：执行

Domain logic：领域逻辑

Data persistence：数据维持

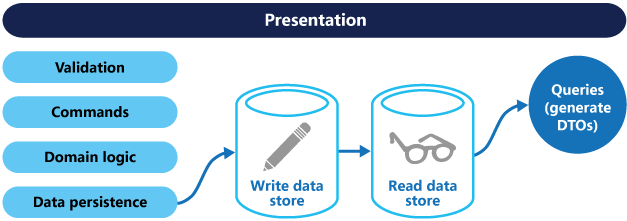
Generate DTOs ：生成

表现层UI（User Interface）

业务逻辑层BLL（Business Logic Layer）

数据访问层（Data Access Layer）

CQRS



主数据库处理CUD，从库处理R，从库的结构可以和主库的结构完全一样，也可以不一样，从库主要用来只读查询操作。在数量上从库的个数也可以根据查询规模进行扩展，在业务逻辑上，也可以根据专题从主库中划分出不同的从库，

