试谈命令查询职责分离(CQRS)模式

胡斌

摘要： 介绍了命令查询职责分离模式，该模式从业务上分离修改和查询的行为，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。简要介绍了传统的CRUD 方式存在的问题、CQRS 模式，并以一个简单的在线日记系统演示了CQRS 模式的实现。

关键词： CRUD 方式；CQRS 模式；Event Sourcing 模式；CQRS 的实现

1 CRUD 方式的问题

在以前的管理系统中，命令(Command，通常用来更新数据，操作DB) 和查询(Query) 通常使用的是在数据访问层中Repository 中的实体对象(这些对象是对DB 中表的映射)，这些实体有可能是SQL Server 中的一行数据或者多个表。

对于一些比较简单的系统，使用这种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地实现功能。但是传统的CRUD 方法有一些问题：

（1） 使用同一个对象实体来进行数据库读写可能会太粗糙，大多数情况下，比如编辑的时候可能只需要更新个别字段，但是却需要将整个对象都穿进去，有些字段其实是不需要更新的。在查询的时候在表现层可能只需要个别字段，但是需要查询和返回整个实体对象。

（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，经常要处理锁的问题，在写入数据的时候，需要加锁。读取数据的时候需要判断是否允许读取。这样使得系统的逻辑性和复杂性增加，并且会对系统吞吐量的增长会产生影响。

（3） 同步地，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。

这里面一个很重要的问题是，系统中的读写频率比，是偏向读，还是偏向写，就如同一般的数据结构在查找和修改上时间复杂度不一样，在设计系统的结构时也需要考虑这样的问题。解决方法就是经常用到的对数据库进行读写分离。让主数据库处理事务性的增、删、改操作(Insert，Update,Delete)，让从数据库处理查询操作(Select)，数据库复制被用来将事务性操作导致的变更同步到集群中的从数据库。这只是从DB 角度处理了读写分离，但是从业务或者系统上面读和写仍然是存放在一起的。他们都是用的同一个实体对象。

要从业务上将读和写分离，就是要谈到介绍的命令查询职责分离模式。

2 CQRS 定义

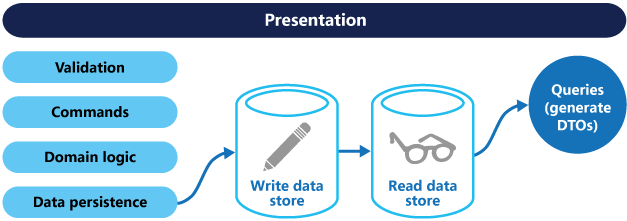
CQRS 最早来自于Betrand Meyer （Eiffel 语言之父，开-闭原则OCP 提出者） 在Object-Oriented Software Construction 这本书中提到的一种命令查询分离(Command Query Separation, CQS) 的概念。其基本思想在于，任何一个对象的方法可以分为两大类：

（1） 命令(Command)： 不返回任何结果(void)，但会改变对象的状态。

（2） 查询(Query)： 返回结果，但是不会改变对象的状态，对系统没有副作用。

操作和查询分离使得人们能够更好地把握对象的细节，能够更好地理解哪些操作会改变系统的状态。CQRS 只是简单地将之前只需要创建一个对象拆分成了两个对象，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(这个和CQS 的定义一致)。

CQRS 使用分离的接口将数据查询操作(Queries) 和数据修改操作(Commands) 分离开来，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。这样读和写逻辑就隔离开来了。使用CQRS 分离了读写职责之后，可以对数据进行读写分离操作来改进性能，可扩展性和安全。如图1 所示。



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，从库主要用来进行只读的查询操作。在数量上从库的个数也可以根据查询的规模进行扩展，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。从库也可以实现成ReportingDatabase，根据查询的业务需求，从主库中抽取一些必要的数据生成一系列查询报表来存储。

3 CQRS 特点

CQRS 模式的优点：

（1） 分工明确，可以负责不同的部分。

（2） 将业务上的命令和查询的职责分离能够提高系统的性能、可扩展性和安全性。并且在系统的演化中能够保持高度的灵活性，能够防止出现CRUD 模式中，对查询或者修改中的某一方进行改动，导致另一方出现问题的情况。

（3） 逻辑清晰，能够看到系统中的哪些行为或者操作导致了系统的状态变化。

（ 4） 可以从数据驱动( Data -Driven) 转到任务驱动(Task-Driven) 以及事件驱动(Event-Driven)。

基于以上优点，在以下场景中，可以考虑使用CQRS 模式：

（1） 当在业务逻辑层有很多操作需要相同的实体或者对象进行操作的时候。CQRS 使得可以对读和写定义不同的实体和方法，从而可以减少或者避免对某一方面的更改造成冲突。

（2） 对于一些基于任务的用户交互系统，通常这类系统会引导用户通过一系列复杂的步骤和操作，通常会需要一些复杂的领域模型，并且整个团队已经熟悉领域驱动设计技术。写模型有很多和业务逻辑相关的命令操作的堆，输入验证，业务逻辑验证来保证数据的一致性。读模型没有业务逻辑以及验证堆，仅仅是返回DTO 对象为视图模型提供数据。读模型最终和写模型相一致。

（3） 适用于一些需要对查询性能和写入性能分开进行优化的系统，尤其是读/写比非常高的系统，横向扩展是必须的。对于系统在将来会随着时间不断演化，有可能会包含不同版本的模型，或者业务规则经常变化的系统。

（4） 需要和其他系统整合，特别是需要和事件溯源EventSourcing 进行整合的系统，这样子系统的临时异常不会影响整个系统的其他部分。

4 CQRS 与Event Sourcing 的关系

在CQRS 中，查询方面，直接通过方法查询数据库，然后通过DTO 将数据返回。在操作(Command) 方面，是通过发送Command 实现，由CommandBus 处理特定的Command，然后由Command 将特定的Event 发布到EventBus 上，然后Event-Bus 使用特定的Handler 来处理事件，执行一些诸如，修改、删除、更新等操作。这里，所有与Command 相关的操作都通过Event 实现。这样可以通过记录Event 来记录系统的运行历史记录，并且能够方便地回滚到某一历史状态。Event Sourcing就是用来进行存储和管理事件的。这里不展开介绍。

5 CQRS 的实现

CQRS 模式在思想上比较简单，但是实现上还是有些复杂。它涉及到DDD，以及Event Sourcing，这里使用一个实例来说明CQRS 模式。这个实例是一个简单的在线记日志系统，实现了日志的增删改查功能。整体结构如图2 所示。

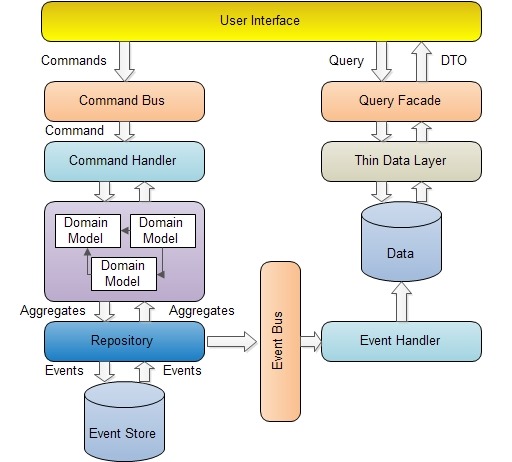


图2 很清晰地说明了CQRS 在读写方面的分离，在读方面，通过QueryFacade 到数据库里去读取数据，这个库有可能是ReportingDB。在写方面，比较复杂，操作通过Command 发送到CommandBus 上，然后特定的CommandHandler 处理请求，产生对应的Event，将Eevnt 持久化后，通过EventBus 特定的EevntHandler 对数据库进行修改等操作。

6 结语

CQRS 是一种思想很简单清晰的设计模式，它通过在业务上分离操作和查询来使得系统具有更好的可扩展性及性能，使得能够对系统的不同部分进行扩展和优化。在CQRS 中，所有的涉及到对DB 的操作都是通过发送Command，然后特定的Command 触发对应事件来完成操作，这个过程是异步的，并且所有涉及到对系统的变更行为都包含在具体的事件中，结合Eventing Source 模式，可以记录下所有的事件，而不是以往的某一点的数据信息，这些信息可以作为系统的操作日志，可以对系统进行回退或者重放。

参考文献

[1] 武海燕, 晏立. 嵌入式实时软件的任务构造[J] . 计算机工程, 2010，（7）.

[2] 王乾宇, 朱小冬, 王毅刚, 周鹏. 嵌入式软件仿真测试环境中实时控制器的设计[J] . 计算机测量与控制, 2012，（1）.

[3] 殷永峰, 刘斌, 王晨. 实时嵌入式软件测试执行引擎的设计与实现[J] . 北京航空航天大学学报, 2010，（6）.

[4] 余耀. 解析计算机软件设计中嵌入式实时软件的应用[J] .中国新技术新产品, 2014，(08).

[5] 马宇驰. 计算机软件设计中嵌入式实时软件的应用探析[J] .信息通信, 2014，(04).